SIEMENS

SIMATIC

STEP 7
From S5 to S7

Converter Manual

Preface, Contents

Part 1: Planning Your Conversion

Introduction 1

Hardware 2

Software 3

Part 2: Converting Programs

Procedure 4

Preparing for Conversion 5

Conversion 6

Editing the Converted Program 7

Compiling 8

Application Example 9

Appendix

Address and Instruction Lists A

Literature List B

Glossary, Index

Edition 12/2002

A5E00171227-01

Safety Guidelines

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger

indicates that death, severe personal injury or substantial property damage will result if proper precautions are not taken.

Warning

indicates that death, severe personal injury or substantial property damage can result if proper precautions are not taken.

Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Note

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified Personnel

The device/system may only be set up and operated in conjunction with this manual.

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage

Note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

SIMATIC®, SIMATIC NET®, and SIMATIC HMI® are registered trademarks of SIEMENS AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Copyright © Siemens AG 1997-2002 All rights reserved

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Siemens AG
Bereich Automation and Drives
Industrial Industrial Automation Systems
Postfach 4848, D-90327 Nuernberg

Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

© Siemens AG 1997-2002
Subject to change without prior notice.

Preface

Purpose of the Manual

This manual supports you when converting S5 programs into S7. With the information in this manual you can do the following:

- Convert existing S5 programs into S7 programs and subsequently edit them manually if necessary.
- Incorporate pre-converted S7 functions (previous S5 standard function blocks) into your S7 programs.

Audience

This manual is intended for programmers who wish to use existing S5 programs in S7.

Where is this Manual Valid?

This manual is valid for release 4.0 of the STEP 7 programming software.

Where Does this Manual Fit in with the Rest of the S7 Documentation?

There is a wide range of user documentation available to support you in configuring and programming an S7 programmable controller which is intended to be used selectively. The following explanations should make it easier for you to use the user documentation.

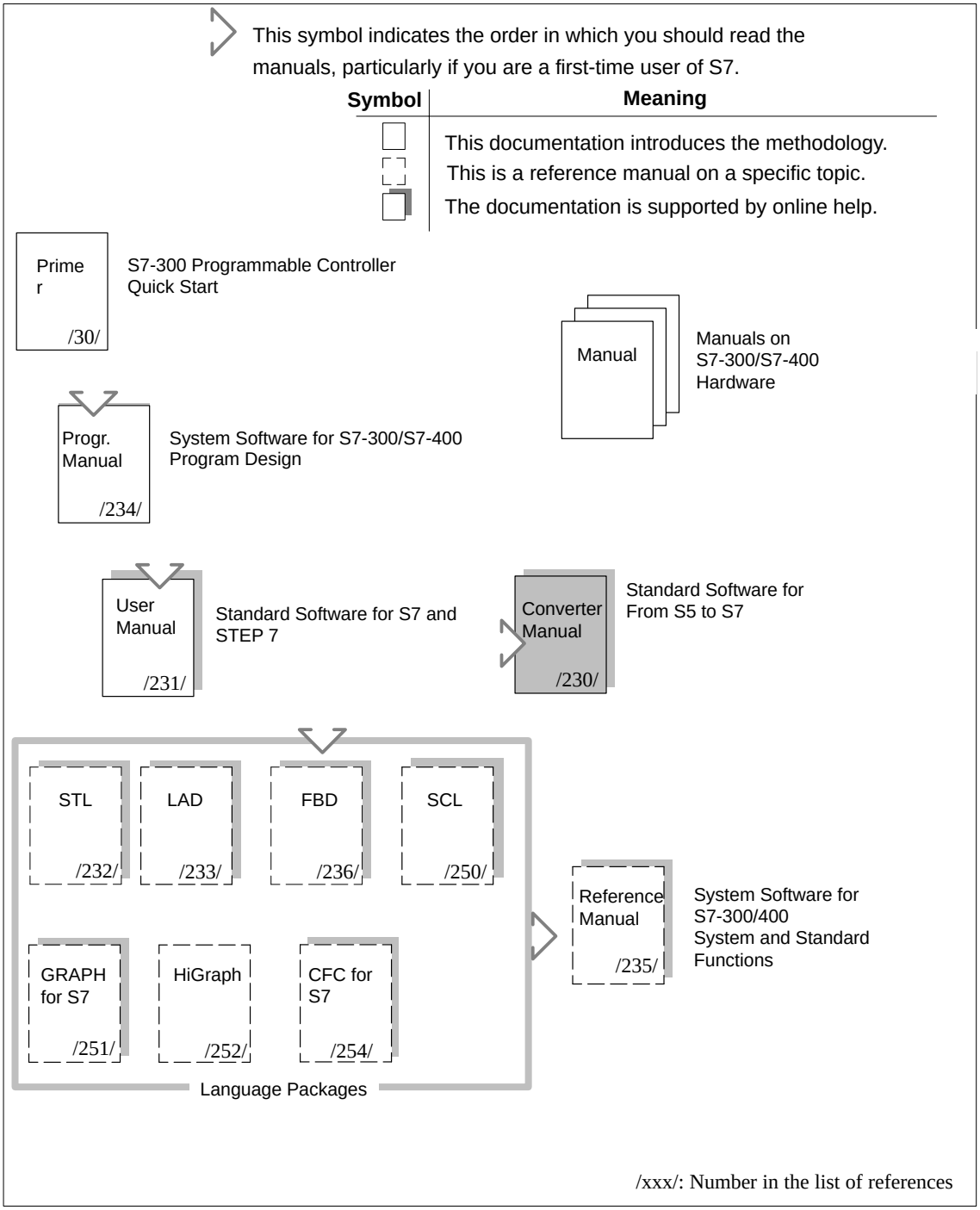

Figure 1-1 S7 Information Landscape

Table 1-1 S7 Documentation Content

Title	Subject
S7-300 Programmable Controller Quick Start, Primer	The *Primer* offers a basic introduction to the methodology of the structure and programming of an S7-300/S7-400. It is especially suited to first-time users of an S7 programmable control system.
S7-300 and S7-400 Program Design Programming Manual	The *S7-300/S7-400 Program Design Programming Manual* provides basic information on the structure of the operating system and of a user program of an S7 CPU. The first-time user of an S7-300 or S7-400 should use this manual to acquire an overview of the programming methodology and to use it to base their user program design on.
S7-300 and S7-400 System and Standard Functions Reference Manual	The S7 CPUs have integrated system functions and organization blocks included with their operating system, which you can use when programming. The manual provides you with an overview of the system functions, organization blocks, and loadable standard functions available in S7, and – in the form of reference information – detailed interface descriptions for their use in your user program.
STEP 7 User Manual	The *STEP 7 User Manual* explains the main usage and the functions of the STEP 7 automation software. As a first-time user of STEP 7 and as an experienced user of STEP 5, this manual will provide you with an overview of the procedures used to configure, program, and start up an S7-300/S7-400. While you are working with the software you can access a range of online help topics which offer detailed support on using the software.
Converter Manual From S5 to S7	You will need the *From S5 to S7 Converter Manual* if you want to convert existing S5 programs to run them on S7 CPUs. The manual provides an overview of the procedures and usage of the Converter; you can find a detailed description of the converter functions in the online help. You will also find the interface descriptions for the converted S7 functions available in the online help. Practical information is also provided on SIMATIC S7 hardware and software.
Statement List, Ladder Logic, Function Block Diagram, SCL[1] Manuals	The manuals for the programming language packages Statement List, Ladder Logic, Function Block Diagram, and SCL (Sequential Control Language) contain both the user's guide and the reference description of the programming language or representation type. You only require one language type for programming an S7-300/S7-400, but you can mix the languages within a project, if required. If you are using a language for the first time, it is recommended that you use the manual to learn about the methodology of creating a program in the chosen language first. While you are working with the software you can access a range of online help topics which offer detailed support on using the respective editors/compilers.
GRAPH[1] , HiGraph[1], CFC[1] Manuals	The languages GRAPH, HiGraph, and CFC (Continuous Function Chart) offer additional methods of programming blocks in the form of sequential controls, state graphs, or charts. The manuals contain both the user's guide and the reference description of the programming language. If you are using a language for the first time, it is recommended that you use the manual to learn about the methodology of creating a program in the chosen language first. While you are working with the software you can access a range of online help topics which offer detailed support on using the respective editors/compilers (with the exception of HiGraph).

1 Optional package for system software for S7-300/S7-400

Structure of the Manual

This manual assumes you have knowledge of S7 programs which you can read about in the *Programming Manual* /**234**/. You should also be familiar with using the Standard software, as described in the *User Manual* /**231**/.

The manual is divided according to the following topic areas:

- Part 1 (Chapters 1 to 3) explains how to plan your conversion from S5 to S7.
- Part 2 (Chapters 4 to 9) explains how to convert programs with the converter.
- Chapter 9 contains application examples.
- The Appendix is a reference section on all STL instructions (international and German mnemonics) provided for reference purposes.
- The Glossary explains important terms.
- The Index will help you to locate text passages on important subjects quickly and easily.

Conventions

References to other manuals are shown using the part number of the literature between slashes /.../. Using these numbers you can find out the exact title of the manual from the literature list at the end of this manual.

Additional Assistance

If you have any questions regarding the software described in this manual and cannot find an answer here or in the online help, please contact the Siemens representative in your area. You will find a list of addresses in the Appendix of /**70**/ or /**100**/, or in catalogs, and in Compuserve (`go autforum`). You can also speak to our Hotline under the following phone or fax number:

Tel. (+49) (911) 895 7000 (Fax 7001)

If you have any questions or comments on this manual, please fill out the remarks form at the end of the manual and return it to the address shown on the form. We would be grateful if you could also take the time to answer the five questions giving your personal opinion of the manual.

Siemens also offers a number of training courses to introduce you to the SIMATIC S7 automation system. Please contact your regional training center or the central training center in Nuremberg, Germany for details:

D-90327 Nuremberg, Tel. (+49) (911) 895-3154.

Notes

This manual replaces the older “Converting S5 Programs” User Manual. In other manuals, this manual may still be referred to under its old name or simply as the Converter Manual.

Contents

1 Introduction 1-1

2 Hardware 2-1

2.1 Programmable Logic Controllers 2-2

2.2 S7 Modules 2-4
2.2.1 Central Processing Units (CPU) 2-6
2.2.2 Power Supply Modules (PS) 2-8
2.2.3 Interface Modules (IM) 2-9
2.2.4 Communications Processors (CP) 2-10
2.2.5 Function Modules (FM) 2-13
2.2.6 Signal Modules (SM) 2-15
2.2.7 Simulation Modules (S7-300) 2-16

2.3 Distributed I/O Devices 2-17

2.4 Communication 2-18
2.4.1 Interface to User Programs 2-20

2.5 Operator Interface 2-21

3 Software 3-1

3.1 General Operating Principles 3-1
3.1.1 Installation Requirements 3-1
3.1.2 Installing STEP 7 Software 3-2
3.1.3 Starting STEP 7 Software 3-3

3.2 Structure of an S7 Project 3-4

3.3 Editing Projects with the SIMATIC Manager 3-7
3.3.1 Creating Projects 3-7
3.3.2 Storing Projects 3-8

3.4 Configuring Hardware with STEP 7 3-9

3.5 Configuring Connections in the Connection Table 3-11

3.6 Inserting and Editing a Program 3-13
3.6.1 Basic Procedure for Creating Software 3-13
3.6.2 Inserting Components for Creating Software in S7 and M7 Programs 3-15

3.7 Blocks 3-17
3.7.1 Comparison 3-17
3.7.2 Functions and Function Blocks 3-18
3.7.3 Data Blocks 3-18
3.7.4 System Blocks 3-19
3.7.5 Organization Blocks 3-20
3.7.6 Block Representation during Conversion 3-24

3.8 System Settings 3-26

3.9 Standard Functions 3-28
3.9.1 Floating-Point Math 3-28
3.9.2 Signal Functions 3-28
3.9.3 Integrated Functions 3-28
3.9.4 Basic Functions 3-29
3.9.5 Analog Functions 3-29
3.9.6 Math Functions 3-29

3.10 Data Types 3-30

3.11 Address Areas 3-32
3.11.1 Overview 3-32
3.11.2 New Addresses in S7: Local Data 3-33

3.12 Instructions 3-35

3.13 Addressing 3-39
3.13.1 Absolute Addressing 3-39
3.13.2 Symbolic Addressing 3-39
3.13.3 New Feature: Complete Addressing of Data Addresses 3-41
3.13.4 Indirect Addressing 3-43

4 Procedure 4-1

4.1 Analyzing the S5 System 4-2

4.2 Creating an S7 Project 4-4

4.3 Configuring Hardware 4-4

5 Preparing for Conversion 5-1

5.1 Providing the Required Files 5-2

5.2 Checking Addresses 5-3

5.3 Preparing the S5 Program 5-4

5.4 Creating Macros 5-5
5.4.1 Instruction Macros 5-6
5.4.2 OB Macros 5-7
5.4.3 Editing Macros 5-8

6 Conversion 6-1

6.1 Starting the Conversion 6-1

6.2 Generated Files 6-5

6.3 Interpreting Messages 6-8

7 Editing the Converted Program 7-1

7.1 Address Changes 7-2
7.1.1 Options for Changing Addressing 7-2

7.2 Non-Convertible Functions 7-3

7.3 Indirect Addressing – Conversion 7-4

7.4 Working with Direct Memory Accesses 7-5

7.5 Assigning Parameters 7-5
7.6 Standard Functions 7-6
8 Compiling the Program 8-1
9 Application Example 9-1
9.1 Analog Value Processing 9-2
9.2 Temporary Local Data 9-5
9.3 Evaluating the Start Information from the Diagnostic Interrupt OB (OB82) 9-8
9.4 Block Transfer 9-11
9.5 Calling the Examples 9-14
A Address and Instruction Lists A-1
A.1 Addresses A-1
A.2 Instructions A-3
B Literature List B-1
Glossary Glossary-1
Index Index-1

Part 1: Planning Your Conversion

Introduction 1

Hardware 2

Software 3

Introduction 1

Until now you were familiar with the name SIMATIC as the synonym for SIEMENS programmable controllers of the S5 family. Now the name SIMATIC stands for fully integrated automation.

The concept **fully integrated automation** describes a revolutionary new way of combining the worlds of manufacturing and process engineering. All hardware and software components are integrated into one single system: SIMATIC.

This complete integration is made possible by the universal compatibility offered by the S7 system in the following three areas:

- Database

  Data are only entered once and are then available to a whole factory. Transfer errors and inconsistencies are therefore a thing of the past.

- Configuring and programming

  All the components and systems belonging to a task are planned, configured, programmed, commissioned, debugged, and monitored with one single fully integrated software package with a modular design - under one user interface and with the most suitable utility.

- Communication

  “Who communicates with whom” is determined simply in a connection table and can be changed at any time. The various network types can be configured easily and uniformly.

To be able to meet the wide range of possibilities of SIMATIC as a fully integrated system, brand new concepts have been shaped in SIMATIC S7. Some functions are therefore achieved in other ways to those you are familiar with in S5.

The STEP 7 programming software is based on new technology and concepts. For example, the user interface is designed to meet modern ergonomic requirements and runs under Windows 95/NT. In our programming languages, we have endeavored to adhere to the IEC 1131 standard as closely as possible without becoming incompatible with STEP 5.

We are convinced that our new STEP 7 system meets the following demands:

- A software basis for fully integrated automation
- Programming which conforms to IEC 1131
- Compatibility with STEP 5

We are also aware that converting from an existing system to a new system gives rise to a number of questions and we recognize that it will be necessary to make certain adaptations, particularly with regard to the software.

This manual is intended to provide answers to these questions and, at the same time, show you simple ways in which you can continue to use your existing STEP 5 programs in SIMATIC S7.

Hardware

2

This chapter describes the hardware that can be used for S7 and makes comparisons, when necessary, with the hardware used for S5, in order to facilitate the transition from S5 to S7.

Converting Hardware from S5 to S7 using the Siemens Catalog on CD-ROM

The Siemens CD-ROM “Components for Automation” / catalog CA01 (from 4/97) contains an application designed to aid you in choosing hardware when converting from S5 to S7. To access the catalog of products, select the menu command **Auswahlhilfen > Simatic**. Here you can enter any S5 system desired; the application uses this system data to create a rack configuration and a signal list. You can then convert this S5 configuration to an S7 configuration.

2.1 Programmable Logic Controllers

SIMATIC S7 consists of the following three types of programmable logic controllers classified according to their performance range:

SIMATIC S7-200 SIMATIC S7-200 is a compact micro programmable logic controller (PLC) designed for applications having the lowest performance range. S7-200 is controlled by its own system-specific software package which is not included in the following comparison of S5 and S7.

SIMATIC S7-300 SIMATIC S7-300 is a modular mini controller designed for applications having a low performance range.

SIMATIC S7-400 SIMATIC S7-400 is designed for applications providing an intermediate to high performance range.

For easy reference, S7-300 module names always start with a “3” and S7-400 module names with a “4”.

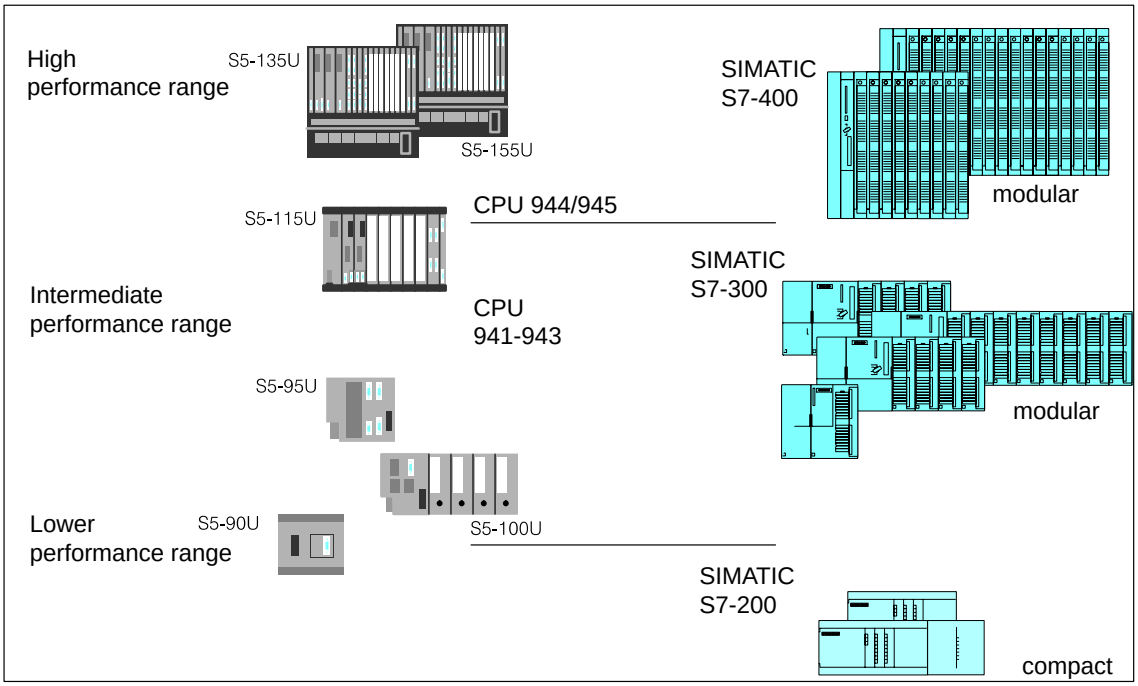

Figure 2-1 SIMATIC Programmable Controllers

Connecting Programming Devices and OPs to SIMATIC S7

Programming device interface MPI (Multipoint Interface) for programming devices and operator panels

The programming device interface AS511 used in SIMATIC S5 has been replaced by the multipoint interface, MPI (for S7-300 and S7-400). This multipoint interface provides a direct electrical connection for HMI devices (HMI: Human Machine Interface, previously known as COROS) and for programming devices to the programming device interface used by SIMATIC S7. The interfaces are completely integrated.

The following table provides a direct comparison of these two interface specifications:

AS511	MPI
25-pin TTY interface (20 mA)	9-pin sub-D interface with RS485 technology
Transmission rate: 9.6 Kbps	Transmission rate: 187.5 Kbps
Protocol: 3964R	Protocol: S7 functions
	Max. length of network: 50 m (with bus amplifiers or special cables: up to 1000 m) All programmable modules within a configuration can be addressed via the multipoint interface
One device can be connected	Up to 31 devices can be connected

Bus interface for Operator Panels (OP)

Programmable logic controllers in the SIMATIC S5 and SIMATIC S7 automation families can be connected using the **PROFIBUS** (previously known as SINEC L2) bus system. As before, these connections are bus-specific.

2.2 S7 Modules

No Fundamental Differences from S5

The range of modules used in S7 corresponds to and expands on the known and proven module concept used in SIMATIC S5.

S7 contains the following types of modules:

- Central processing units (CPU)
- Power supply modules (PS)
- Interface modules (IM)
- Communications processors (CP); (such as for connecting to PROFIBUS)
- Function modules (FM); (such as for counting, positioning, closed-loop control)
- Digital and analog modules are now called “signal modules” (SM)

This chapter describes the similarities and differences in the range of modules used in SIMATIC S5 and SIMATIC S7.

New Performance Characteristics

S7 modules can be distinguished by the following features:

- The new modules are not equipped with jumpers or switches.
- The new modules do not require cooling fans. As in S5, they have the IP 20 protection class.
- The new modules can be assigned parameters and have diagnostic capability.
- The S7 slot assignment is more flexible than for S5.
- Expansion devices and ET 200 distributed I/O devices can trigger interrupts.

Comparison of Module Parameter Assignment in S5 and S7

The following table compares the module parameter assignment in SIMATIC S5 and SIMATIC S7:

SIMATIC S5	SIMATIC S7
	Modules are arranged (hardware configuration) using the STEP 7 application for configuring hardware
Addresses are set with DIL switches	Addresses are set with the STEP 7 application for configuring hardware or are slot-oriented
System behavior is set with DIL switches	Module parameters are assigned with the STEP 7 application for configuring hardware
CPU parameters for operational behavior are assigned via system data areas or DB1 / DX0	CPU parameters are assigned with the STEP 7 application for configuring hardware
	Compiled configuration data are downloaded to the CPU; Module parameters are transferred automatically on startup

2.2.1 Central Processing Units (CPU)

S7-300 CPUs

Table 2-1 lists the most important performance specifications for S7-300 CPUs. If you want to replace an S5 CPU, you can use the following table to compare performance in order to select the most suitable CPU:

Table 2-1 Performance Features of the S7-300 CPUs

Feature	CPU 312 IFM	CPU 313	CPU 314	CPU 314 IFM	CPU 315	CPU 315-2 DP
Work memory (integrated)	6 Kbytes	12 Kbytes	24 Kbytes	24 Kbytes	48 Kbytes	
Load memory						
• integrated	20 Kbytes RAM; 20 Kbytes EEPROM	20 Kbytes RAM	40 Kbytes RAM	40 Kbytes RAM; 40 Kbytes EEPROM	80 Kbytes RAM	
• expandable with memory card	–	up to 512 Kbytes	up to 512 Kbytes	–	up to 512 Kbytes (in CPU programmable up to 256 Kbytes)	
Process image size, inputs and outputs	32 bytes + 4 on-board	128 bytes	128 bytes	124 bytes + 4 on-board	128 bytes	
I/O address area						
• digital inputs/outputs	Inputs: 128 + 10 on-board Outputs: 128 + 6 on-board	128	512	Inputs: 496 + 20 on-board Outputs: 496 + 16 on-board	1024	
• analog inputs/outputs	32		64	Inputs: 64 + 4 on-board Outputs: 64 + 1 on-board	128	
Bit memory	1024	2048				
Counters	32	64				
Timers	64	128				
Max. sum of all retentive data	72 bytes		4736 bytes	144 bytes	4736 bytes	
Local data	512 bytes in total; 256 bytes per priority class	1536 bytes in total; 256 bytes per priority class				
Blocks:						
OBs	3	13	13	13	13	14
FBs	32	128	128	128	128	128
FCs	32	128	128	128	128	128
DBs	63	127	127	127	127	127
SFCs	25	44	48	48	48	53
SFBs	2	7	7	14	7	7

S7-400 CPUs

The CPUs for the S7-400 have different performance features. Table 2-2 shows a comparison of the performance features of these CPUs.

Table 2-2 Performance Features of the S7-400 CPUs

Feature	CPU 412-1	CPU 413-1	CPU 413-2 DP	CPU 414-1	CPU 414-2 DP	CPU 416-1	CPU 416-2 DP
Work memory (integrated)	48 Kbytes	72 Kbytes		128 Kbytes	128/384 Kbytes	512 Kbytes	0.8/1.6 Mbytes
Load memory • integrated • expandable with memory card	8 Kbytes up to 15 Mbytes			8 Kbytes up to 15 Mbytes		16 Kbytes up to 15 Mbytes	
Process image size, inputs and outputs	128 bytes each			256 bytes each		512 bytes each	
I/O address area • digital inputs/outputs max. • analog inputs/outputs max.	2 Kbytes 16384 1024			8 Kbytes 65536 4096		16 Kbytes 131072 8192	
Bit memory	4096 M 0.0 to M 511.7			8192 M 0.0 to M 1023.7		16384 M 0.0 to M 2047.7	
Counters	256 C 0 to C 255			256 C 0 to C 255		512 C 0 to C 511	
Timers	256 T 0 to T 255			256 T 0 to T 255		512 T 0 to T 511	
Local data	4 Kbytes in total			8 Kbytes in total		16 Kbytes in total	
Blocks: OBs FBs FCs DBs SFBs SDBs	23 256 256 511 24 512			31 512 1024 1023 24 512		44 2048 2048 4095 24 512	
SFCs	55	55	58	55	58	55	58

Retentive Features of S7-400

The CPUs for SIMATIC S7-400 require a backup battery to buffer timers, counters, and bit memory.

Retentive Features of S7-300 without Backup Battery

The CPUs for S7-300 do not require a battery to buffer timers, counters, or bit memory. Similarly, the contents of data blocks can also be retained in the event of a power failure. The CPUs for SIMATIC S7-300 have a maintenance-free backup that saves those addresses and data which have parameters specifying that they be retained in event of a power failure.

The size and quantity of the available retentive areas depend on the respective CPU.

Parameter Assignment for the Retentive Feature

The size of the data retention areas is set in parameter assignment dialog boxes during hardware configuration with STEP 7.

2.2.2 Power Supply Modules (PS)

A selection of power supply modules is available for each programmable logic controller (PLC).

Power Supply Modules for S7-300

Any 24-volt power source (industrial) can be used to supply to the CPU in S7-300.

The range of modules in S7 contains the following power supplies specifically designed for S7-300:

Module Name	Output Current	Output Voltage	Input Voltage
PS 307	2 A	24 VDC	120 / 230 VAC
PS 307	5 A	24 VDC	120 / 230 VAC
PS 307	10 A	24 VDC	120 / 230 VAC

Power Supply Modules for S7-400

Module Name	Output Current	Output Voltage	Input Voltage
PS 407 4A	4 A 0.5 A	5 VDC 24 VDC	120 / 230 VAC
PS 407 10A	10 A 1 A	5 VDC 24 VDC	120 / 230 VAC
PS 407 20A	20 A 1 A	5 VDC 24 VDC	120 / 230 VAC
PS 405 4A	4 A 0.5 A	5 VDC 24 VDC	24 VDC
PS 405 10A	10 A 1 A	5 VDC 24 VDC	24 VDC
PS 405 20A	20 A 1 A	5 VDC 24 VDC	24 VDC

For further information, see the Reference Manuals /**71**/ and /**101**/.

2.2.3 Interface Modules (IM)

Some interface modules in S5 have been replaced in S7. This change primarily affects local area connections. In S7, it is recommended that PROFIBUS be used to transmit signals for remote area connections.

Comparison of IM Modules

S5 Module	S7-300 Module	S7-400 Module	Description
IM 305 IM 306 IM 300 / IM 312	IM 365 IM 360 / IM 361	IM 460-0 / IM 461-0 IM 460-1 / IM 461-1	Central configuration
–	–	IM 460-3 / IM 461-3	Remote area (up to 100 m)
IM 301 / IM 310	Connection via PROFIBUS	Connection via PROFIBUS	Connection of I/O modules and signal preprocessing modules (up to 200 m)
IM 304 / IM 314	Connection via PROFIBUS	Connection via PROFIBUS	Use of distributed I/O in remote areas (up to 600 m)
		IM 463-2	Distributed connection of S5 expansion devices in remote areas (up to 600 m)
IM 307 / IM 317	Connection via PROFIBUS	Connection via PROFIBUS	Connection via fiber-optic cable (up to 1500 m)
IM 308 / IM 318	Connection via PROFIBUS	Connection via PROFIBUS	Distances up to 3000 m

In S7, the interface module IM 467 can be used in place of IM 308C.

You can use the interface module IM 463-2 to connect S5 digital and analog modules to the S7 mounting rack with IM 314 via an S5 expansion rack.

Connectable S5 Expansion Racks

The following S5 expansion racks can be connected:

- EG 183 expansion unit
- EG 185 expansion unit
- ER 701-2
- ER 701-3

2.2.4 Communications Processors (CP)

The following section lists the S5 and S7 communications processors that can be used in various subnets. In addition, the services supported by these processors are also indicated.

Subnets in SIMATIC

In order to meet the varying requirements of different automation levels (such as on the processing, cell, field, and actuator-sensor levels), SIMATIC provides the following subnets:

- **AS Interface**

  The actuator-sensor interface (AS-i) is a connection system for the lowest processing level in automation systems. It is primarily used for networking binary sensors and actuators. Its data quantity is limited to a maximum of 4 bits per slave.

- **MPI**

  The multipoint interface (MPI) subnet is intended for short-range field and cell levels. The MPI is a multipoint interface used in SIMATIC S7/M7 and C7. It is designed as a programming device interface and is intended for networking a small number of CPUs and for exchanging small quantities (up to 70 bytes) of data.

- **PROFIBUS**

  PROFIBUS is the network used for the cell and field areas in open, manufacturer-independent, SIMATIC communication systems. PROFIBUS is suitable for quick transmission of moderate quantities of data (approx. 200 bytes).

- **Industrial Ethernet**

  Industrial Ethernet is the network used for the processing and cell levels in open, manufacturer-independent, SIMATIC communication systems. Industrial Ethernet is suitable for quick transmission of large quantities of data.

- **Point-To-Point Connection**

  A point-to-point connection is not a subnet in the traditional sense. This connection is established in SIMATIC by using point-to-point communications processors (CP) to connect two communication partners (such as PLCs, scanners, PCs) with each other.

AS Interface (SINEC S1)

The following table provides an overview of the modules available for communicating via the actuator-sensor (AS) interface.

S5 Module	S7-300 Module	S7-400 Module
CP 2433 (AS-i functions) CP 2430 (AS-i functions)	CP 342-2 (AS-i functions)	–

MPI (SINEC L1)

Communication via SINEC L1 in S5 has been converted to global data communication using MPI in S7.

All CPUs in S7-300 and S7-400 as well as the programming devices and operator panels have an MPI interface.

PROFIBUS (SINEC L2)

The following table provides an overview of the modules available for communicating with PROFIBUS and which services are supported by these modules.

S5 Module	S7-300 Module	S7-400 Module
CP 5431 (FMS, FDL, DP) CPU 95U (FDL, DP *))	CP 342-5 (S7 functions, FDL, DP) CP 343-5 (S7 functions, FDL, FMS)	CP 443-5 Ext. (S7 functions, FDL, DP) CP 443-5 Basic (S7 functions, FDL, FMS)
IM 308-B/C (DP)	CPU 315-2 DP (DP)	CPU 413-2 DP (DP) CPU 414-2 DP (DP) CPU 416-2 DP (DP) IM 467 (DP)

*) depends on the specific equipment ordered

Industrial Ethernet (SINEC H1)

The following table provides an overview of the modules available for communicating with Industrial Ethernet and indicates which services are supported by these modules.

S5 Module	S7-300 Module	S7-400 Module
CP 1430 TF (ISO transport)	CP 343-1 (S7 functions, ISO transport)	CP 443-1 (S7 functions, ISO transport)
CP 1430 TCP (ISO on TCP)	CP 343-1 TCP (S7 functions, ISO on TCP)	CP 443-1 TCP (S7 functions, ISO on TCP)

Point-To-Point Connection

The following table provides an overview of the modules available for point-to-point connection and which services are supported by these modules.

S5 Module	S7-300 Module	S7-400 Module
CP 521 (3964(R), ASCII) CP 523 (3964(R), ASCII)	CP 340-RS 232C (3964(R), ASCII) CP 340-20 mA (3964(R), ASCII) CP 340-RS 422/485 (3964(R), ASCII)	CP 441-1 (3964(R), RK512, ASCII)
CP 544 (3964(R), RK 512, ASCII)	–	
CP 524/525 (3964(R), RK 512, ASCII, additional special drivers which can be loaded CP 544 B (3964(R), RK 512, ASCII, additional special drivers which can be loaded	–	CP 441-2 (3964(R), RK512, ASCII, additional special drivers which can be loaded

2.2.5 Function Modules (FM)

Some IP and WF modules in SIMATIC S5 can be used in S7-400 with the help of a special adapter casing. In other cases, there are new function modules available in S7 to help you obtain the functionality desired.

The following table provides an overview and comparison of the signal preprocessing modules available in S5 and S7.

Table 2-3 Comparison of Signal-Preprocessing Modules in S5 and S7

S5 Module	Adapter Casing	S7 Module	Description
IP 240	yes	FM 451 (limited)	Counter, position detection, and positioning modules
IP 241	no	FM 451 / FM 452 (limited)	Digital position detection module
IP 242A	no	no	Counter module
IP 242B	yes	no	Counter module
IP 244	yes	FM 455	Controller module
IP 246I/A	yes	FM 354 / FM 357 / FM 453	Positioning module for variable speed drives
IP 247	yes	FM 353 / FM 357 / FM 453	Positioning module for stepper motors
IP 252	no	FM 455 (limited)	Closed-loop control module
IP 260	no	FM 355 (limited)	Closed-loop control module
IP 261	no	no	Proportioning module
IP 281	no	FM 350-1 / FM 450-1	Counter module
IP 288	no	FM 451 / FM 452	Positioning module for regulating rapid/creep feed and cam control
WF 705	yes	FM 451 (limited)	Position detection module
WF 706	no	FM 451 (limited)	Positioning and counter module
WF 707	no	FM 452 (limited)	Cam control
WF 721	yes	FM 354 (limited because of assembly technology)	Positioning module

Table 2-3 Comparison of Signal-Preprocessing Modules in S5 and S7, continued

S5 Module	Adapter Casing	S7 Module	Description
WF 723A	yes	FM 453	Positioning module
WF 723 B	yes	FM 357 (limited because of assembly technology)	Positioning module
WF 723 C	yes	no	Positioning module
–	–	FM 456-4	Application module (M7-FM)
–	–	SINUMERIK FM-NC	Numeric control
–	–	FM STEPDRIVE	Stepper motor control
–	–	SIMOSTEP	Stepper motor

2.2.6 Signal Modules (SM)

The signal modules in SIMATIC S7 are comparable in function to the input/output modules in S5. However, in addition to simple signal modules, S7 also provides modules that can be assigned parameters and which have diagnostic capability.

Signal Modules which can be Assigned Parameters

Digital input modules in S7 that can be assigned parameters allow you to specify (with the STEP 7 application for configuring hardware) which channels are to trigger a hardware interrupt on edge change.

The input ranges of analog input modules can be easily assigned parameters with STEP 7.

Signal Modules with Diagnostic Capability

Modules with diagnostic capability can detect both external errors such as wire breaks or external short circuits and internal ones such as RAM errors or short circuits within modules.

A diagnostic event is processed by the controller in the following two ways:

- By triggering a diagnostic interrupt. This notifies the appropriate organization block (OB) in the user program, which then interrupts the cyclic program.
- By making an entry in the diagnostic buffer of the CPU, which can then be read with a programming device or operator interface device.

The following tables list the signal modules available in S7:

Table 2-4 Signal Modules in SIMATIC S7-300

DI (SM 321)	DO (SM 322)	AI (SM 331)	AO (SM 332)
32 x 24 VDC	32 x 24 VDC/0.5 A	8 x 12 bit	2 x 12 bit
16 x 24 VDC	16 x 24 VDC/0.5 A	2 x 12 bit	
16 x 24 VDC with hardware and diagnostic interrupt	8 x 24 VDC/0.5 A with diagnostic interrupt	Ex: 4 x 15 bit	Ex: 4 x 15 bit
16 x 24 VDC M-reading	8 x 24 VDC/2 A	Ex: 12 x 15 bit	
8 x 120/230 VAC	8 x 120/230VAC / 2 A	AI 4/AO 2 X 8/8 bit (SM 334)	
Ex: 4 x 24 VDC	Ex: 4 x 15 VDC/ 20m A		
	Ex: 4 x 24 VDC/ 20m A		

Table 2-5 Signal Modules in SIMATIC S7-400

DI (SM 421)	DO (SM 422)	AI (SM 431)	AO (SM 432)
32 x DC 24 V	32 x 24 VDC/0.5 A	8 x 13 bit	8 x 13 bit
16 x 24/60 VUC with hardware and diagnostic interrupt	16 x 24 VDC/2 A	8 x 14 bit (for temperature measurement)	
16 x 120/230 VUC	16 x 120/230 VAC /5 A	8 x 14 bit	
32 x 120 VUC	16 x 120/230 VAC /2 A	16 x 16 bit	
	16 x 30/230 VUC/ Rel. 5 A		

2.2.7 Simulation Modules (S7-300)

S7-300 provides a simulation module, SM 374, for testing your program.

This simulation module has the following capabilities:

- It can simulate
  - 16 inputs,
  - 16 outputs, or
  - 8 inputs and 8 outputs (each having the same initial address)
- Its functions can be set with a screwdriver
- It can provide status displays for simulating inputs or outputs

2.3 Distributed I/O Devices

The modules for distributed I/O devices in the ET 200 system which already existed in SIMATIC S5 can continue to be used in SIMATIC S7.

In addition, there are new ET 200 modules to extend the range.

DP Masters

The following modules can be a **DP master** in the distributed I/O system:

- S7-300 with CPU 315-2 DP or CP 342-5 as DP master
- S7-400 with CPU 413-2 DP / 414-2 DP / 416-2DP or CP 443-5, extended as DP master

DP Slaves

The following are examples of devices which can be **DP slaves** in the distributed I/O system:

- Distributed I/O devices such as ET 200B, ET 200C, ET 200M, ET 200X (up to 12 Mbps) and ET 200U, ET 200L (up to 1.5 Mbps)
- Programmable logic controllers such as
  - S5-115U, S5-135U, or S5-155U with IM 308-C as DP slave
  - S5-95U with DP slave interface (up to 1.5 Mbps)
  - S7-300 with CPU 315-2 DP or CP 342-5 as DP slave
  - S7-400 with CP 443-5 as DP slave
- Interface to actuator-sensor interface with the DP/AS-i link
- Text displays and operator panels for machine-like operator control and monitoring
- MOBY identification systems
- Low-voltage switching devices
- Field devices (such as drives, valve islands, etc.) from Siemens or other manufacturers.

FMS Masters

The following can serve as an **FMS master**:

- S7-300 with CP 343-5 as FMS master
- S7-400 with CP 443-5 Basic as FMS master

FMS Slaves

Examples of devices that can serve as an **FMS slave** are the ET 200U or the SIMOCODE motor protection and control device.

For further information, refer to the appropriate manuals or the Siemens catalog CA01.

2.4 Communication

Services and Subnets

Communication within SIMATIC S7 is based on different subnets which provide various services.

Services	S7 Communication Functions (S7 Functions)		
	ISO transport ISO-on-TCP	FDL (SDA) FMS DP	GD
Subnets	Industrial Ethernet	PROFIBUS	MPI

The following is a summary of the communication services used in SIMATIC:

S7 Functions

The S7 functions provide services for communicating between S7/M7 CPUs, S7 OP/OSs and PCs. These S7 functions are already integrated in each SIMATIC S7/M7 device. Since these S7 functions correspond to a service in the ISO application layer, they are independent of any one subnet and can thus be used on all subnets (MPI, PROFIBUS, Industrial Ethernet).

ISO Transport

These functions are used for secure data transfer from SIMATIC S7 to SIMATIC S5.

They are used to transfer moderate amounts of data (up to 240 bytes) via open communication at ISO transport layer 4 based on the ISO reference model for Industrial Ethernet.

ISO on TCP

These functions are used for secure data transfer from SIMATIC S7 to SIMATIC S5.

They are used to transfer moderate amounts of data (up to 240 bytes) via open communication according to the TCP/IP protocol at ISO transport layer 4 based on the ISO reference model for Industrial Ethernet.

The ISO-on-TCP service requires the extended RFC1006 standard.

FDL (SDA)

These functions are used for secure data transfer from SIMATIC S7 to SIMATIC S5.

They are used to transfer moderate amounts of data (up to 240 bytes) via open communication at Fieldbus Data Link (FDL) layer 2 based on the ISO reference model for Industrial Ethernet.

FMS

PROFIBUS FMS (Fieldbus Message Specification) provides services for transferring structured data (FMS variables) over static FMS connections.

The FMS service can be classified at layer 7 of the ISO reference model. It corresponds to the European standard EN 50170 Vol. 2 PROFIBUS and provides services for transferring structured data (variables).

DP

PROFIBUS DP services allow transparent communication with distributed I/O devices. These distributed I/O devices are addressed by the control program in exactly the same manner as centralized I/O devices.

GD

Global Data Communication is a simple communication option integrated in the operating system of S7-300/400 CPUs.

GD communication permits cyclic exchange of data between CPUs via the multipoint interface; for S7-400, it also allows event-driven data exchange.

2.4.1 Interface to User Programs

The communication interface to a user program consists of the following blocks:

- SFCs (without connection configuration)
- SFBs (with connection configuration) (only S7-400)
- Loadable FCs / FBs

These blocks replace the S5 handling blocks. The functionality here is similar, but it is now accomplished using STEP 7 languages. To establish communication, you will have to adapt an appropriate S5 program with handling functions to the new blocks.

Network	**Service**	**Interface in S5 User Program**	**Interface in S7 User Program**
Point-to-point connection	–	Handling blocks *	S7-300: loadable FBs S7-400: loadable SFBs
PROFIBUS	FDL (PLC-PLC) Free Layer 2 FMS	Handling blocks * Handling blocks * Handling blocks *	Loadable FCs – Loadable FBs
Industrial Ethernet	ISO 4 ISO 4 + AP STF MAP	Handling blocks * Handling blocks * Handling blocks * + loadable FBs Handling blocks * + loadable FBs	Loadable FCs – – Loadable FBs

* Integrated or loadable, depending on CPU

2.5 Operator Control and Monitoring

Introduction

The following section provides an overview of the extent to which SIMATIC HMI (HMI: Human Machine Interface, previously COROS) operator panels can be used in SIMATIC S7.

Operator Panels

The SIMATIC HMI operator panel provides operator control and monitoring functions for SIMATIC S5, SIMATIC S7, and SIMATIC TI, as well as for other controllers.

STEP 5

In general, a standard function block, which is called depending on the operator panel connected, is required in the programmable controller for connecting **SIMATIC OP to SIMATIC S5**.

The following operator panels (OP) can be used with S5:

- TD17, OP5/A1, OP7/PP, OP7/DP-12, OP15/x1, OP17/PP, OP17/DP-12
- OP25, OP35, OP37, TP37

STEP 7

When connecting **SIMATIC OP to SIMATIC S7/M7**, a distinction must be made between PPI, MPI, and PROFIBUS as MPI nodes.

PPI or MPI connections run via the programming device interface in the CPU. In doing so, SIMATIC OP uses the communication services of SIMATIC S7/M7 (S7 functions); this means that a standard function block is not required.

The PROFIBUS connection from SIMATIC OP to SIMATIC S7/M7 also involves communication accomplished using S7 functions. Again, this means that a standard function block is not required. (SIMATIC OPs are “active nodes” and not PROFIBUS-DP slaves as is the case for the PROFIBUS connection to SIMATIC S5.) The same number of nodes that applies to an MPI connection also applies here.

The following operator panels (OP) can be used with S7:

- TD17, OP3, OP5/A2, OP7/DP, OP7/DP-12, OP15/x2, OP17/DP, OP17/DP-12
- OP25, OP35, OP37, TP37

The following restrictions apply to SIMATIC OPs:

- OP3: up to 2 connections
- OP5/15/25: up to 4 connections
- TD17, OP7/17: up to 4 connections
- OP35: up to 6 connections
- OP37, TP37: up to 8 connections

Configuration

SIMATIC ProTool and SIMATIC ProTool/Lite are modern tools for configuring operator panels. SIMATIC ProTool can be used to configure all devices, while SIMATIC ProTool/Lite can only be used to configure line-oriented operator panels. Functionally, ProTool/Lite is a subset of ProTool.

Integration in SIMATIC STEP 7

ProTool can be integrated in the SIMATIC STEP 7 configuration software; this enables direct access to configuration data such as symbol tables and communication parameters used for control configuration. This feature not only saves time and money; it also prevents errors resulting from redundant data entry.

Table 2-6 Configuration Tools for Operator Interface Devices

Device	Configuration Tool
Line-oriented OP (TD17, OP3, OP5, OP7, OP15, OP17)	ProTool/Lite or ProTool
Graphic-oriented OP (OP25, OP35, OP37, TP37)	ProTool

WinCC

WinCC can be used for a single or multi-terminal (client-server arrangement) system.

WinCC is a system for creating solutions to visualization and process control tasks used in production and process automation; it is compatible with all business sectors and technologies. It provides function modules suitable for displaying graphics and messages, archiving information, and record-keeping in industrial applications. Its powerful and efficient hardware connection, quick display updating, and secure data archiving provide users with high flexibility and availability.

In addition to these system functions, WinCC also provides open interfaces for creating user-specific solutions. These allow WinCC to be integrated in complex, company-wide automation solutions. Integrated features allowing access to data archives via standard interfaces such as ODBC and SQL and integration of objects and documents via OLE2.0 and OLE Custom Controls (OCX) are also included. These mechanisms make WinCC an effective communication partner for Windows applications.

WinCC is based on the 32-bit operating systems MS Windows 95 or MS Windows NT. Both feature preemptive multitasking, which ensures quick reaction to process events and provides a high level of security against data loss. Windows NT provides additional security functions and can also serve as the basis for server operation in a WinCC multi-terminal system. The WinCC software is itself a 32-bit application which was developed using the most modern object-oriented software technology.

Software 3

3.1 General Operating Principles

Overview

The software for configuring and programming SIMATIC S7/M7/C7 is designed according to modern ergonomic concepts and is thus largely self-explanatory.

3.1.1 Installation Requirements

Operating System

Microsoft Windows 95.

Standard Hardware

Programming device or PC with the following specifications and equipment:

- A 80486 processor (or higher)
- A minimum 16 Mbytes RAM (32 Mbytes recommended)
- A VGA monitor or other type of monitor supported by Windows 95
- A keyboard, and optional but recommended, a mouse supported by Microsoft Windows 95

Storage Capacity

The following storage space is required on the hard disk:

- The Standard package with one language installed occupies 105 Mbytes on the hard disk. The exact amount of space required depends on the amount of standard software installed.
- STEP 7 should also have about 64 Mbytes of total memory available for storing swap files. For example, if you have 32 Mbytes of RAM, you will need an additional 32 Mbytes of virtual memory.
- Approx. 50 Mbytes should be available for user data.
- A minimum of 1 Mbyte free memory should be available on the hard disk for setup. (The setup files are deleted once the installation is complete.)

3.1.2 Installing the STEP 7 Software

Overview

STEP 7 contains a setup program that carries out the installation automatically. User prompts appearing on the screen guide you step-by-step through the entire installation procedure.

Authorization

A product-specific user authorization is required to use the STEP 7 programming software. Software protected in this manner can only be used if the required authorization for the program or software package is located on the hard disk of the respective programming device or PC.

To obtain this authorization, you need the copy-protected authorization diskette included in the consignment. This diskette also contains the program AUTHORS, which is required to display, install, and uninstall STEP 7.

The procedure for transferring and removing this authorization is described in the *User Manual* **/231/**.

Note

Siemens programming devices (such as the PG 740) are supplied with installable STEP 7 software already on the hard disk.

For further information on installing STEP 7, see the *User Manual* **/231/**.

3.1.3 Starting the STEP 7 Software

Starting

After starting Windows 95/NT you will find the icon for the SIMATIC Manager on the Windows user interface. This is the access point to the STEP 7 software.

Double-clicking the “SIMATIC Manager” icon is the fastest way to start STEP 7. This icon opens the window for the SIMATIC Manager. From here, you can access the standard system, all optional software, and all functions that you have installed.

Alternatively, you can also start the SIMATIC Manager by clicking the “Start” button in the Windows 95/NT taskbar. The menu title for this is found under “Simatic/STEP 7.”

SIMATIC Manager

The SIMATIC Manager is the initial window used for configuring and programming. Here you can do the following:

- Set up projects
- Configure and assign parameters to hardware
- Configure communication connections
- Create programs
- Test programs and start them running

Access to functions is object-oriented, intuitive, and easy to learn.

You can work with the SIMATIC Manager in the following ways:

- Offline (not connected to a controller), or
- Online (connected to a controller)

(When doing this, be sure to observe the appropriate safety guidelines.)

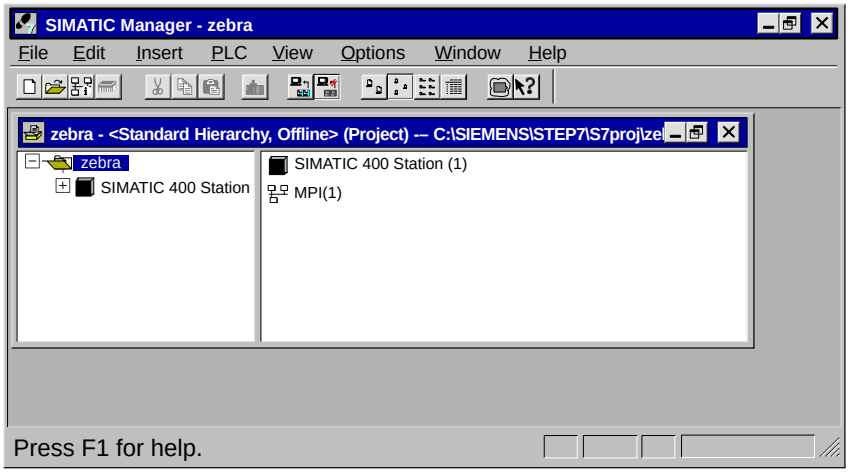

Figure 3-1 SIMATIC Manager with an Open Project

3.2 Structure of an S7 Project

Definition

Projects contain all the data and programs for an automation solution. Their purpose is to provide organized storage of data and programs created for such an application.

Projects in STEP 5

You will already be familiar with the term “project” from working with STEP 5. In STEP 5, a project contains all **STEP 5** files created for **one** user program in a project file.

This project file contains information necessary for convenient editing and maintenance of a user program, such as parameter settings, as well as catalog and file names.

Projects in STEP 7

In STEP 7, a project contains all the programs and data necessary for an automation solution, regardless of the number of CPUs involved and how they are networked. Thus, a project is not just limited to a user program used for a particular programmable module; instead, it contains several user programs used for many programmable modules, which are all stored together under a common project name.

Note

As in STEP 5, it is also possible in STEP 7 to create a simple user program intended for only one CPU. In this case, a project is limited to one CPU.

The following section discusses the directory structure that STEP 7 provides for the user programs and data that you create.

Components of a Project

A project in STEP 7 essentially consists of the objects depicted in Figure 3-2. These objects are listed and explained below.

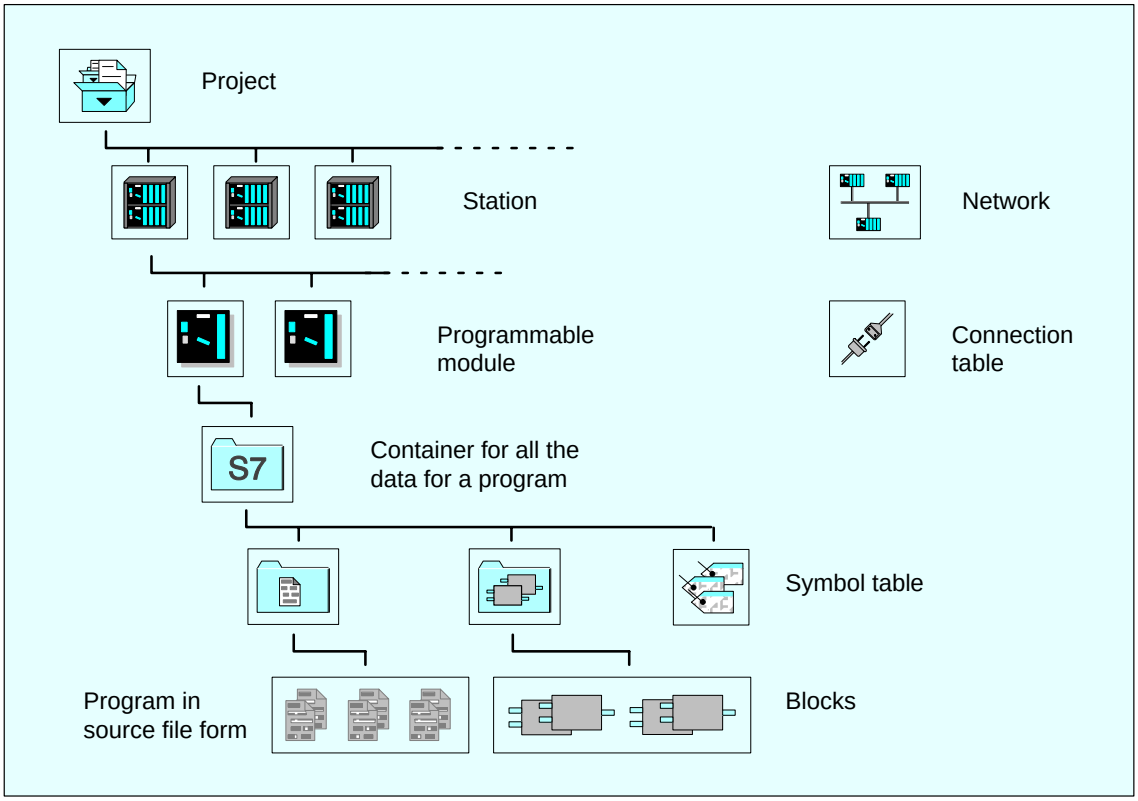

Figure 3-2 Basic Objects in a STEP 7 Project with their Hierarchical Structure

Network

The “Network” object contains the definitive properties for a subnet such as MPI or PROFIBUS. Assigning a station or a communication module within a station to a network enables STEP 7 to check communication parameters for consistency.

Station

A station represents the structure of a programmable controller along with all the racks belonging to it. If a module with a DP interface is part of a station, then the entire master system (that is, the DP slaves belonging to it) is also part of this station.

A station consists of one or more programmable modules, such as a CPU.

Hardware

Hardware is an object containing the configuration data and parameters for a station. The configuration data and parameters for a station are stored in system data blocks (SDBs).

Programmable Modules

In contrast to other modules, programmable modules contain user programs. In the folders (known as “containers” in STEP 7) found in the programmable modules you will find all the data belonging to the program for the module. Examples of such programs are the following:

- Programs in source file form (created with a text editor)

  When the source program is compiled, executable blocks are created in the “Blocks” container.

- Blocks which are loaded into the programmable module
- Symbol tables

Connection Table

The connection table depicts all connections for a programmable module, such as a CPU, in a station. A connection defines the communication properties between two nodes and is identified by a connection ID. This connection ID is all that you need to program event-controlled communication using standardized communication blocks, which are similar to the handling blocks found in STEP 5.

Source Files

In S7 programming, source files are the basis for creating blocks. Source files cannot be downloaded to an S7 CPU.

Blocks

Blocks are distinct parts of a user program and are distinguished by their function, structure, and use within it. Blocks can be downloaded to S7 CPUs.

In addition to the executable blocks, the “Blocks” container also contains the variable tables.

Symbol Table

The symbol table shows the assignment of symbolic names, for example, for inputs, outputs, bit memory, and blocks.

3.3 Editing Projects with the SIMATIC Manager

3.3.1 Creating Projects

New Project

To create a project, follow the steps outlined below:

1. Select the menu command **File ▸ New** in the SIMATIC Manager.
2. Select the option “New Project” in the “New” dialog box.
3. Enter a name for the project and confirm your entry with “OK.”

Alternative Procedures

When editing a project, you are flexible as to the order in which you perform most of the tasks. Once you have created a project, you can choose one of the following methods:

- First configure the hardware and then create the software for it, or
- Start by creating the software independent of any configured hardware. The hardware configuration of a station does not need to be established before entering a program.

Table 3-1 Alternative Procedures

Alternative 1	Alternative 2
Configure the hardware first (see also Section 3.4)	**Create the software first**
Configure your hardware (see Section 3.4).	
Once the configuration is complete, the “S7 Program” containers required for creating software are already inserted and available.	Insert the required software containers (S7 Programs) in your project (see Section 3.6).
Then create the software for the programmable modules (see Section 3.6).	Then create the software for the programmable modules (see Section 3.6).
	Configure your hardware (see Section 3.4).
	Link the S7 program to a CPU once you have configured the hardware.

The procedure for downloading and testing your program without a hardware configuration is described in the *User Manual* **/231/**.

3.3.2 Storing Projects

Overview

To back up a project, you can save a copy of the project under another name or archive it.

Save As...

To save the project under another name, proceed as follows:

1. Open the project.
2. Select the menu command **File ▸ Save As**. The “Save As” dialog box is displayed.
3. Select either save with or without a consistency check and close the dialog box with “OK.” The “Save As” dialog box is displayed.
4. Under “Save In,” select the directory in which the project is to be saved.
5. In the “File Name” field, enter a file name in place of the asterisk (*). Do not change the file extension.
6. Close the dialog box with “OK.”

Make sure that there is enough memory available on the drive selected. For example, it is not advisable to select a disk drive to back up a project because a project is generally too large to fit on a diskette. You must archive projects before saving them on diskettes. Archives can then be split up over several diskettes.

Archiving

You can store individual projects or libraries in compressed form in an archive file located on a hard disk or a transportable data medium (diskette).

In order to be able to access components of an archived project or library, the project must first be extracted from the archive. Archiving is discussed in detail in the *User Manual* ***/231/***.

3.4 Configuring Hardware with STEP 7

SIMATIC S5 did not provide an option for configuring hardware using the software. In S7, addressing and assigning parameters to modules and configuring communications is carried out by means of a STEP 7 application. The advantage of this method is that the user no longer has to make any settings on the modules, since the configuring and assigning of parameters can now be done centrally from the programming device.

Prerequisite

To configure hardware, a project must already have been created.

Inserting a Station

To create a new station in a project, open the project to display the project window (if this has not already been done).

1. Select the project.
2. Create the object for the required hardware by selecting the menu command **Insert ▸ Station**.

In the submenu you can select one of the following options:

- SIMATIC 300 station
- SIMATIC 400 station
- PC/programming device
- SIMATIC S5
- Other stations, meaning non-SIMATIC S7/M7 and SIMATIC S5

The stations PC/programming device, SIMATIC S5, and other stations are only listed for configuring communication links. Configuration and programming of S5 stations is not possible.

Click on the “+” sign in front of the project icon in the project window if the station is not displayed below it.

Configuring the Hardware

To configure the hardware, proceed as follows:

1. Click the new station you have inserted; it contains the “Hardware” object.
2. Open the “Hardware” object. The “HWConfig” window is displayed.
3. In the “Hardware Configuration” window, plan the structure of the station. A catalog of modules is available to help you do this. If this is not already displayed, select the menu command **View ▸ Catalog** to view it.
4. Insert a rack from the module catalog in the empty window. Then select the modules and place them in the rack slots. At least one CPU must be configured for each station. During this procedure, HWConfig automatically checks all entries you make.

For further information on configuring hardware, see the *User Manual* **/231/**.

Result of Configuration

For each CPU you create in your configuration, an S7 program and a connection table (“Connections” object) are created automatically once you have saved and exited the hardware configuration. The S7 program contains the “Source Files” and “Blocks” objects as software containers as well as the symbol table.

The “Blocks” container contains the object for OB1 and the “System Data” object with the compiled configuration data.

3.5 Configuring Connections in the Connection Table

In S5, connections are configured with COM NCM. There is a COM package for each communications processor (CP). In S7, all connections are configured in the connection table.

Overview

Configuring connections is a prerequisite for using SFB communication functions in a user program.

A connection determines the following:

- The communication partners involved in the S7 project,
- The type of connection established (such as an S7 connection or FDL connection),
- Special properties such as the active or passive establishment of a connection or whether operating mode messages are to be sent.

When you configure connections, a unique local identifier (known as the local ID) is issued for each connection. This local ID is all you require when assigning communication parameters.

Each CPU that can serve as the end point of a connection has its own connection table.

Special Feature

If both communication partners are S7-400 stations, a local ID is automatically issued for both end points of the connection. Only one local ID is generated on the S7-400 station for connections to an S7-300 station.

Loading Configuration Data

The local configuration data for connection end points on an S7 station must be separately downloaded into each target station.

An (empty) connection table (“Connections” object) is automatically created for each CPU. This connection table is used to define communication links between CPUs in a network. After this is opened, a window is displayed containing a table for defining connections between programmable modules (For more information about defining connections, see the *User Manual* ***/231/****).*

Example: Connection to an S5 Device

This example shows you how to configure a connection to a SIMATIC S5 station. It assumes that you have already inserted a SIMATIC 400 station in your project.

- Insert a SIMATIC S5 station in your project and then set the properties of the station.
- Open the connection table for the S7 station and select the menu command **Insert ▸ Connections** to insert a connection. A dialog box is displayed in which you can enter the communication partner (the SIMATIC S5 station) and the type of connection.
- Once you have entered this information, the connection appears in the connection table. The properties for the connection must be entered in the corresponding COM NCM for the S5 station.

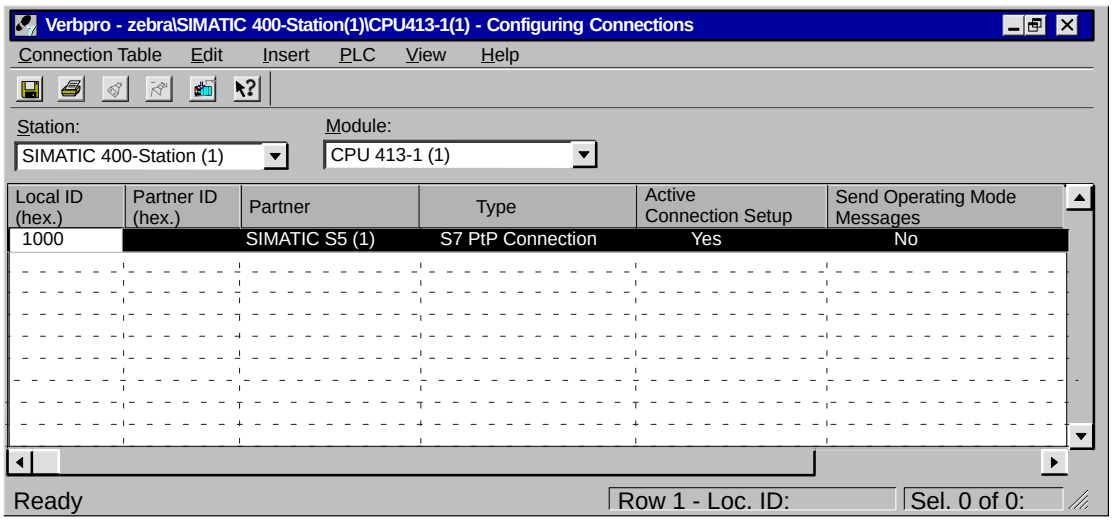

Figure 3-3 Connection Table

3.6 Inserting and Editing a Program

The procedure described in this section applies to creating a new program.

3.6.1 Basic Procedure for Creating Software

Overview

The software for CPUs is stored in program containers. In SIMATIC S7 modules, such an object is called an “S7 Program.”

The figure below shows an S7 program in the CPU of a SIMATIC 300 station.

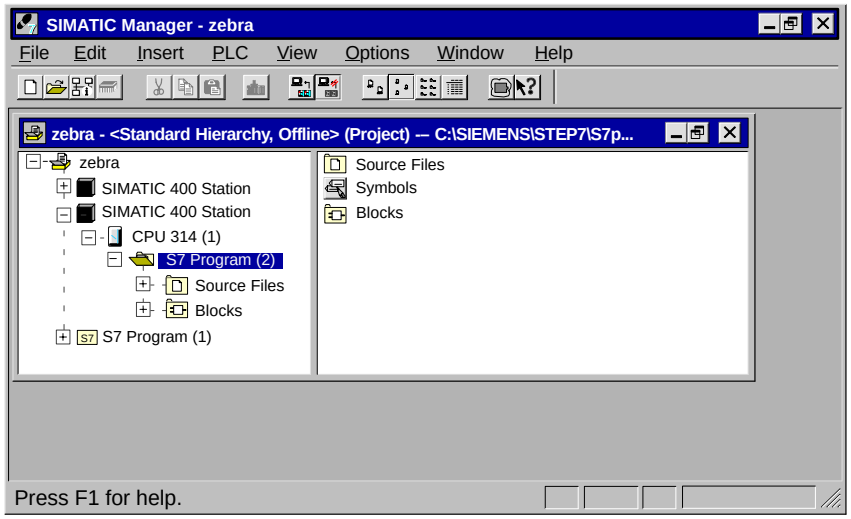

Figure 3-4 Open S7 Program in the SIMATIC Manager

Procedure

To create the software for your project, proceed as follows:

1. Open the S7 program.
2. Open the “Symbols” object in the S7 program and define the symbols. (This step can also be done later.) You will find more information on defining symbols in Section 3.13.2.
3. Open the “Blocks” container if you want to create blocks, or open the “Source Files” container if you want to create a source file.
4. Insert a block or a source file, as appropriate. (For detailed information, see Section 3.6.2). The following menu commands are used for this:
   – **Insert ▸ S7 Software ▸ Block**, or
   – **Insert ▸ S7 Software ▸ Source File**
5. Open the block or source file and enter a program. You will find more information on programs in the Programming Manuals /**232**/-/**236**/.

Depending on your task, you may not need to perform all these steps.

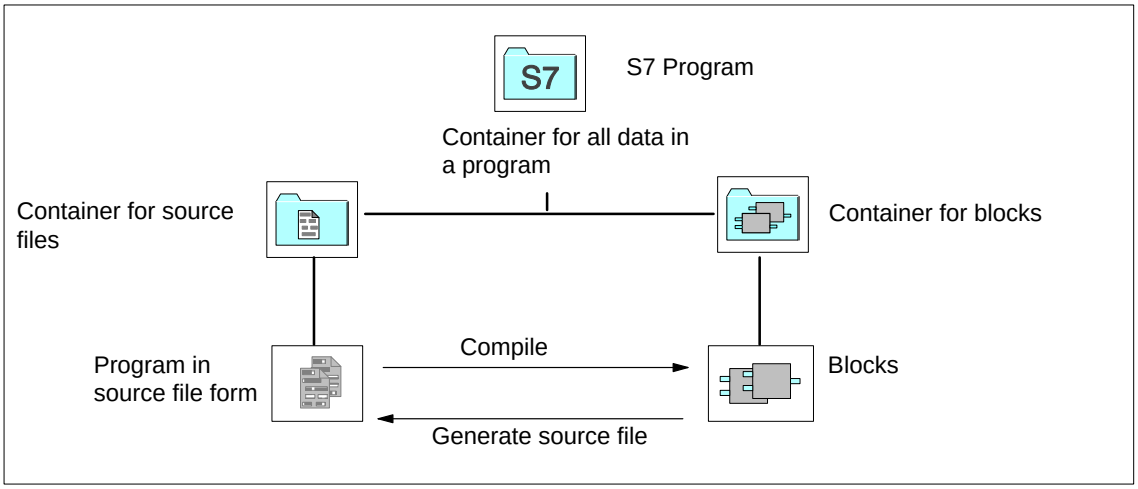

Figure 3-5 Basic Objects in a STEP 7 Project and Their Hierarchical Structure

3.6.2 Inserting Components for Creating Software in S7 and M7 Programs

Existing Components

An S7/M7 program is created automatically for each programmable module as a container for the software:

The following objects already exist in a new S7 program:

- Symbol table (“Symbols” object)
- A “Blocks” container for blocks with OB1 as the first block
- A “Source Files” container for programs in the form of source files

Creating S7 Blocks

If you want to create Statement List, Function Block Diagram, or Ladder Logic programs, select the existing “Blocks” object and then click the menu command **Insert ▸ S7 Software ▸ Block**. In the submenu, you can select the type of block you want to create (such as a data block, user-defined data type (UDT), function, function block, organization block, or variable table (VAT)).

You can now open the (empty) block and start entering the Statement List, Ladder Logic, or Function Block Diagram program. You will find more information in the *Statement List* **/232/**, *Ladder Logic* **/233/**, and *Function Block Diagram* **/236/** Programming Manuals.

The “System Data” object (SDB) which may exist in a user program was created by the system. You can open it to view its contents, but you cannot make changes to it for reasons of consistency. It is used to make changes to the configuration once you have loaded a program and to download the changes to the programmable controller.

Using Blocks from Standard Libraries

You can also use blocks from the standard libraries supplied with the software to create user programs. You access the libraries using the menu command **File ▸ Open**. You will find more information on using standard libraries and creating your own libraries in the online help.

Creating Source Files

If you want to create a source file in Statement List, select the “Source Files” or “Charts” object in the S7 program and then select the menu command **Insert ▸ S7 Software ▸ Source File**. In the submenu, you can select the source file which matches your programming language. You can now open the empty source file and start entering your program.

Creating a Symbol Table

An (empty) symbol table (“Symbols” object) is created automatically when the S7 program is created. When you open the symbol table, the “Symbol Editor” window opens displaying a symbol table where you can define symbols (refer to Section 3.13.2 for more details).

Inserting External Source Files

You can create and edit source files with any ASCII editor. You can then import these files into your project and compile them into executable blocks. To insert an external source file, proceed as follows:

1. Select the “Source Files” container to which you want to import the source file.
2. Select the menu command **Insert ▸ External Source File**.
3. Enter the source file name in the dialog box which appears.

The blocks created when the imported source file is compiled are stored in the “Blocks” container.

3.7 Blocks

3.7.1 Comparison

The following table provides a comparison of the blocks in STEP 5 and STEP 7. The table is intended to answer the question “Which STEP 7 block should I use for which STEP 5 block?”

No Fixed Assignment

This table is not to be interpreted as a fixed one-to-one set of assignments since the new block environment makes additional programming options available. The table entries are to be understood as a set of recommendations for starting STEP 7 programming.

Table 3-2 Comparison of Blocks: STEP 5 / STEP 7

STEP 5 Block	STEP 7 Block	Explanation
Organization block (OB)	Organization blocks (OB)	Interface to the operating system
Integrated special OBs	System functions (SFC) System function blocks (SFBs)	System functions in STEP 7 replace the special organization blocks (STEP 5) that can be called in the user program.
Function block (FB, FX)	Function (FC)	Functions (FCs) in STEP 7 have the same properties as function blocks in STEP 5.
Program block (PB)	Function block (FB)	Program blocks correspond to the function blocks in STEP 7. Function blocks in STEP 7 have completely new properties compared to blocks in STEP 5 having the same name; thus, they now provide new programming options. Note: During conversion, program blocks are transformed into functions (FCs).
Sequence block (SB)	–	There are no sequence blocks in STEP 7.
Data block (DB, DX)	Data block (DB)	In STEP 7 the data blocks are longer than in STEP 5 (in S7-300 up to 8 Kbytes, in S7-400 up to 64 Kbytes).
Data block DX0, DB1 in its special function	System data blocks (SDB) (CPU parameter assignment)	The new system data blocks contain all the hardware configuration data, including the CPU parameter assignments, which determine the program processing.
Comment blocks DK, DKX, FK, FKX, PK	–	In STEP 7 there are no longer any comment blocks. Comments are contained in the respective block in the offline database.

3.7.2 Functions and Function Blocks

Functions (FCs)

Functions (FCs) are logic blocks without a “memory”. The output parameters contain the calculated function values after the function is processed. It is then up to the user how the actual parameters are used and saved after the FC is called.

Do **not** confuse functions with function blocks! In STEP 7 these are different types of blocks.

Function Blocks (FBs)

Function blocks (FB) are logic blocks which do have a “memory.” The memory is in the form of an instance data block which is associated with the function, in which the actual parameters and static data of the function block are stored.

Function blocks are used for applications such as programming controller structures.

3.7.3 Data Blocks

Data blocks store the data for the user program. A distiction is made between shared data blocks and instance data blocks, as explained in the following:

- Shared data blocks are not assigned to any particular block (as in STEP 5).
- Instance data blocks are associated with a function block (FB) and contain, in addition to the FB data, the data from multiple instances that may have been defined.

Every data block can either be a shared data block or an instance data block.

3.7.4 System Blocks

System Functions (SFCs) and System Function Blocks (SFBs)

You do not have to program every function yourself. You can also program communication functions, for example, by using pre-configured blocks that are available in the operating system on the CPUs. These are the following:

- **System functions (SFCs)**, with properties like those of functions (FCs)
- **System function blocks (SFBs)**, with properties like those of function blocks (FBs).

System Data Blocks (SDB)

The previous discussion was centered around blocks containing programs or data from the user program. In addition to these blocks there are other blocks containing settings such as module parameters or addresses. These are called **system data blocks (SDBs)** and are created by special STEP 7 applications, for example, when entering the hardware configuration or creating connection tables.

3.7.5 Organization Blocks

Organization blocks (OBs) form the interface between the operating system and the user program. Different organization blocks carry out their own specific tasks.

Distribution of Organization Blocks

You assemble the STL user program for your S7 CPU from the organization blocks (OBs) required for your automation solution.

Table 3-3 Comparison of the OBs in S5 and S7

Function		S5	S7
Main program	Free cycle	OB1	OB1
Interrupts	Time-delay (delayed) interrupt	OB6	OB20 to OB23
	Time-of-day (clock-controlled) interrupt	OB9	OB10 to OB17
	Hardware interrupts	OB2 to OB5	OB40 to OB47
	Process interrupts	OB2 to OB9 (IB 0)	Replaced by hardware interrupts
	Cyclic (timed) interrupts	OB10 to OB18	OB30 to OB38
	Multicomputing interrupt	–	OB60
Startup	Manual complete (cold) restart	OB21 (S5-115U) OB20 (from S5-135U)	OB100
	Manual (warm) restart	OB21 (from S5-135U)	OB101
	Automatic (warm) restart	OB22	OB101
Errors	Error	OB19 to OB35	OB121, OB122, OB80 to OB87
Other	Processing in STOP mode	OB39	Omitted
	Background processing	–	OB90

Error Handling

Error OBs

Error OBs are called if an error occurs during program execution. You can use them to help program error reactions. If no error OB exists for a particular error type, then the CPU goes into STOP mode.

Table 3-4 Error Handling in S5 and S7

Function	S5	S7
Calling a block which is not loaded	OB19	OB121
Timeout with direct access to I/O modules	OB23	OB122
Timeout updating the process image and the IPC flags (interprocessor communication flags)	OB24	OB122
Addressing error	OB25	OB122
Cycle time exceeded	OB26	OB80
Substitution error	OB27	Omitted
Stop by operator	OB28 (S5-135U)	Omitted
Timeout with input byte IB 0	OB28 (S5-155U)	OB85
Illegal instruction code	OB29 (S5-135U)	STOP
Timeout with direct access to I/Os in the extended address area	OB29 (S5-155U)	OB122
Illegal parameter	OB30 (S5-135U)	Omitted
Parity error or timeout accessing user memory	OB30 (S5-155U)	OB122
Special function group error	OB31	Omitted
Load and transfer error with a data block	OB32	OB121
Collision of timed interrupts	OB33	OB80
Controller error	OB34 (S5-135U)	Omitted
Error generating a data block	OB34 (S5-155U)	SFC feedback
Communication error	OB35	OB84

Troubleshooting in S5 and S7

Exceeded Signal

As in S5, you can also use the status word bits OV and OS to evaluate a report of an exceeded signal. The difference in behavior in the two systems is minor.

For further information about the behavior of status bits with reference to instructions, see the *Statement List Programming Manual* /**232**/.

Integrated Special Functions

The interface between the user program and the system program in S5 CPUs consists of accesses made to the operating system area and via special OBs.

In S7 CPUs, this interface has two new block types ("system functions" and "system function blocks"), in addition to the organization blocks.

System Functions/System Function Blocks

System functions (SFCs) and system function blocks (SFBs) are blocks integrated in the CPU operating system which can be called in a STEP 7 user program as needed. If an error occurs during processing of a system function (SFC), this error can be evaluated in the user program with the aid of the RET_VAL return value.

Table 3-5 Special Functions in S5 and S7

Function	S5 Block	Replacement in S7
Cycle time triggering	OB31	SFC43 RE_TRIGR
Battery failure	OB34	OB81 (Error reaction can be programmed by user)
Access to condition code byte	OB110	STEP 7 instruction: L STW/T STW
Delete ACCU 1 – 4	OB111	STEP 7 instruction sequence: L 0; PUSH; PUSH; PUSH
Roll up ACCU	OB112	Function not identical: STEP 7 instruction: PUSH
Roll down ACCU	OB113	Function not identical STEP 7 instruction: POP
Disable all interrupts on/off	OB120	SFC41 DIS_AIRT SFC42 EN_AIRT
Disable cyclic (timed) interrupts individually on/off	OB121	SFC39 DIS_IRT SFC40 EN_IRT
Delay all interrupts on/off	OB122	SFC41 DIS_AIRT SFC42 EN_AIRT
Delay cyclic (timed) interrupts individually on/off	OB123	SFC39 DIS_IRT SFC40 EN_IRT
Set/read CPU time *(continued on next page)*	OB150	SFC0 SET_CLK SFC1 READ_CLK

Table 3-5 Special Functions in S5 and S7, continued

Function	S5 Block	Replacement in S7
Set/read time-controlled interrupt time	OB151	SFC28 SET_TINT SFC30 ACT_TINT SFC31 QRY_TINT
Cycle statistics	OB152	Local data in OB1
Counter loop	OB160 - 163 (S5-135U)	STEP 7 instruction: LOOP
Variable timed loop	OB160 (S5-115U)	SFC47 WAIT
Read block stack	OB170	Omitted
Variable data block access	OB180	Omitted
Test data block	OB181	SFC24 TEST_DB
Copy data area	OB182	SFC20 BLKMOV
Transfer flags to data blocks	OB190, OB192	SFC20 BLKMOV
Transfer data blocks to flag areas	OB191, OB193	SFC20 BLKMOV
Functions for multiprocessor communication	OB200 - 205	Omitted
Page access	OB216 - 218	No page addressing in S7
Sign extension	OB220	S7 instruction: ITD
Set cycle monitoring time	OB221	Parameter assignment with S7
Restart cycle monitoring time	OB222	SFC43 RE_TRIGR
Compare startup types	OB223	Multicomputing startup only for same startup type
Transfer IPC flags in blocks	OB224	Omitted
Read word from the system program	OB226	Omitted
Read CRC of the system program	OB227	Omitted
Read status information of a program processing level	OB228	SFC51 RDSYSST SFC6 RD_SINFO
Functions for handling blocks	OB230 - 237	Communication with SFBs
Initialize shift register	OB240	Omitted
Process shift register	OB241	Omitted
Delete shift register	OB242	Omitted
Control: Initialize PID algorithm Control: Process PID algorithm	OB250 OB251	Closed-loop control FBs: FB41 - FB43 or SFB41 - SFB43
Transfer data blocks (DB/DX) to DB RAM	OB254, OB255	Omitted

3.7.6 Block Representation during Conversion

Block Assignment

.The block structure has been changed for S7. The figure below shows a simplified example of a block assignment for STEP 5 and STEP 7 resulting from the conversion process.

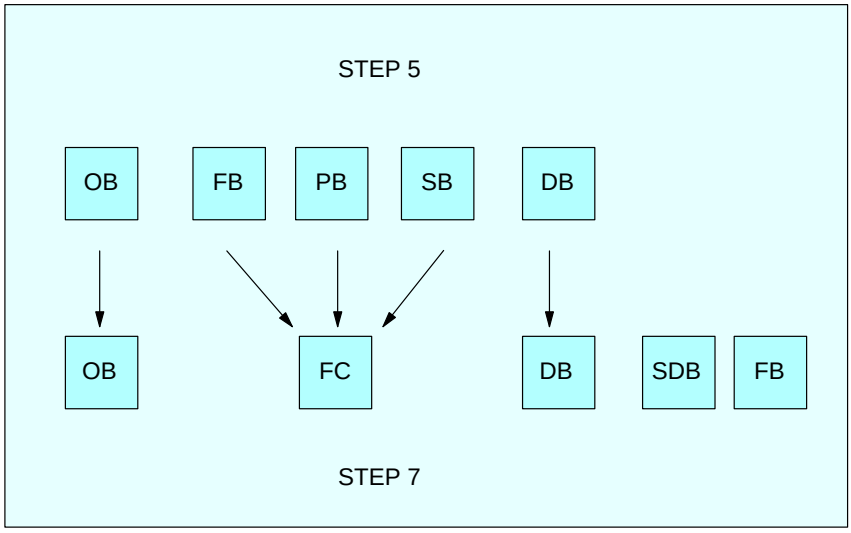

Figure 3-6 Blocks with Comparable Function in STEP 5 and STEP 7

Table 3-6 on the following page shows you how block calls are converted.

Table 3-6 Block Types in S5 and S7

S5			S7	
OB	Fixed numbers	User program	Corresponding S7 OB	Fixed numbers
OB	Fixed numbers	Special function	Not convertible, must be reprogrammed with S7	
PB	0 to 255	User program	FC blocks without parameters	Number is proposed
FB/FX	0 to 255	User program	FC blocks with parameters whose names are retained	Number is proposed
FB	Fixed numbers	Integrated function blocks	Loadable FCs contained in the FBlib1 library which must be loaded to the converted file before compiling	Fixed numbers
FB/FX	Fixed names	Standard function blocks	Loadable FCs contained in the FBlib1 library which must be loaded to the converted file before compiling	Fixed numbers
SB	0 to 255	User program	FC blocks without parameters (sequencers cannot be converted and must be created in S7 GRAPH.	Number is suggested
DB	2 to 255	User data	Shared data blocks (DBs)	Number taken from S5
DX	1 to 255	User data	Shared data blocks (DBs)	Number from 256 onwards is suggested
DB1/ DX0		Data blocks with system settings	If the blocks contain CPU-specific entries, the parameter settings must be made with STEP 7. The converted contents of DB1 and DX0 are irrelevant and can be deleted.	

3.8 System Settings

Converting DB1 and DX0

The following tables show how the functions for the parameters in DB1 and DX0 (system settings) are made:

Table 3-7 Converting the System Settings from DB1

S5 Parameter Block	How Implemented in S7
Startup delay	Call SFC47 WAIT
IPC flags	Set using global data communication, call SFC60 GD_SND SFC61 GD_RCV
Location of error code	System enters error messages in the diagnostic buffer. Information about “Location of error code” omitted
Replace number of integrated FBs	Omitted
On-board analog inputs	Set in HWConfig using CPU properties
On-board interrupt	Set in HWConfig using CPU properties
On-board counter	Set in HWConfig using CPU properties
Change priorities of OBs	Set in HWConfig using CPU properties
Output/disable process image	Call SFC27 UPDAT_PO
Read in/disable process image	Call SFC26 UPDAT_PI
Retentive flags	Set in HWConfig using CPU properties
Retentive timers	Set in HWConfig using CPU properties
Retentive counters	Set in HWConfig using CPU properties
SINEC L1	Replaced by MPI bus (global data communication)
SINEC L2	Set with HWConfig
Software protection	In preparation
Clock parameters	Set in HWConfig using CPU properties or by calling SFC28 SET_TINT
Assigning parameters to timed interrupt OBs	Set in HWConfig using CPU properties
Cycle time monitoring	Set in HWConfig using CPU properties

Table 3-8 Converting the System Settings from DX0

S5 Parameter Block	How Implemented in S7
Addressing error monitoring	Call OB121
Updating the IPC flags	Global data communication
Startup types after power on	Set in HWConfig using CPU properties
Start synchronization in multiprocessor operation	Set in HWConfig using CPU properties
Number of timer cells	Fixed CPU-specific value (for S7-300) or set in HWConfig using CPU properties (for S7-400)
Error handling	Call: SFC36 MSK_FLT SFC37 DMSK_FLT
Floating-point math	Present
Process (hardware) interrupt triggering	Set in HWConfig using CPU properties
Timed (cyclic) interrupt processing mode	Call SFC28 SET_TINT
Cycle time monitoring	Set in HWConfig using CPU properties

3.9 Standard Functions

During conversion, the standard functions present in S5 are automatically replaced by converted functions having the same functionality. In S7, most of these functions can be replaced by simplified command sequences, which conserves memory and reduces the cycle time.

The standard functions are contained in the “StdLib30” S7 library located in the program container FBLib1.

For further information on working with libraries, refer to the online help.

3.9.1 Floating-Point Math

STEP 5	STEP 7		STEP 5	STEP 7	
FB Name	**Number**	**Name**	**FB Name**	**Number**	**Name**
GP:FPGP	FC61	GP_FPGP	GP:MUL	FC65	GP_MUL
GP:GPFP	FC62	GP_GPFP	GP:DIV	FC66	GP_DIV
GP:ADD	FC63	GP_ADD	GP:VGL	FC67	GP_VGL
GP:SUB	FC64	GP_SUB	RAD:GP	FC68	RAD_GP

3.9.2 Signal Functions

STEP 5	STEP 7		STEP 5	STEP 7	
FB Name	**Number**	**Name**	**FB Name**	**Number**	**Name**
MLD:TG	FC69	MLD_TG	MLD:EZ	FC75	MLD_EZ
MELD:TGZ	FC70	MELD_TGZ	MLD:ED	FC76	MLD_ED
MLD:EZW	FC71	MLD_EZW	MLD:EZWK	FC77	MLD_EZWK
MLD:EDW	FC72	MLD_EDW	MLD:EDWK	FC78	MLD_EDWK
MLD:SAMW	FC73	MLD_SAMW	MLD:EZK	FC79	MLD_EZK
MLD:SAM	FC74	MLD_SAM	MLD:EDK	FC80	MLD_EDK

3.9.3 Integrated Functions

STEP 5	STEP 7	
FB Name	**Number**	**Name**
COD:B4	FC81	COD_B4
COD:16	FC82	COD_16
MUL:16	FC83	MUL_16
DIV:16	FC84	DIV_16

3.9.4 Basic Functions

STEP 5	STEP 7		STEP 5	STEP 7	
FB Name	**Number**	**Name**	**FB Name**	**Number**	**Name**
ADD:32	FC85	ADD_32	REG:LIFO	FC93	REG_LIFO
SUB:32	FC86	SUB_32	DB:COPY	FC94	DB_COPY
MUL:32	FC87	MUL_32	DB:COPY	FC95	DB_COPY
DIV:32	FC88	DIV_32	RETTEN	FC96	RETTEN
RAD:16	FC89	RAD_16	LADEN	FC97	LADEN
REG:SCHB	FC90	REG_SCHB	COD:B8	FC98	COD_B8
REG:SCHW	FC91	REG_SCHW	COD:32	FC99	COD_32
REG:FIFO	FC92	REG_FIFO			

3.9.5 Analog Functions

STEP 5	STEP 7		STEP 5	STEP 7	
FB Name	**Number**	**Name**	**FB Name**	**Number**	**Name**
AE:460	FC100	AE_460_1	AE:466	FC106	AE_466_1
AE:460	FC101	AE_460_2	AE:466	FC107	AE_466_2
AE:463	FC102	AE_463_1	RLG:AA	FC108	RLG_AA1
AE:463	FC103	AE_463_2	RLG:AA	FC109	RLG_AA2
AE:464	FC104	AE_464_1	PER:ET	FC110	PER_ET1
AE:464	FC105	AE_464_2	PER:ET	FC111	PER_ET2

3.9.6 Math Functions

STEP 5	STEP 7		STEP 5	STEP 7	
FB Name	**Number**	**Name**	**FB Name**	**Number**	**Name**
SINE	FC112	SINE	ARCCOT	FC119	ARCCOT
COSINE	FC113	COSINE	LN X	FC120	LN_X
TANGENT	FC114	TANGENT	LG X	FC121	LG_X
COTANG	FC115	COTANG	B LOG X	FC122	B_LOG_X
ARCSIN	FC116	ARCSIN	E^X	FC123	E_H_N
ARCCOS	FC117	ARCCOS	ZEHN^X	FC124	ZEHN_H_N
ARCTAN	FC118	ARCTAN	A2^A1	FC125	A2_H_A1

3.10 Data Types

STEP 7 uses new data formats. The table below compares the different data types in S5 and S7:

Table 3-9 Data Types in S5 and S7

Data Types in S5	Data Types in S7	Data Class
BOOL, BYTE, WORD, DWORD, Integer, Double integer, Floating point, Time value, - ASCII character	BOOL, BYTE, WORD, DWORD, INT, DINT, REAL, S5TIME, TIME, DATE; TIME_OF_DAY, CHAR	Elementary data types
–	DATE_AND_TIME, STRING, ARRAY, STRUCT	Complex data types
Timers, Counters, Blocks – –	TIMER, COUNTER, BLOCK_FC, BLOCK_FB, BLOCK_DB, BLOCK_SDB, POINTER, ANY	Parameter types

Table 3-10 Different Formats for Constants in S5 and S7

Formats in S5	Example	Formats in S7	Example
KB	L KB 10	k8	L B#16# A
KF	L KF 10	k16	L 10
KH	L KH FFFF	16#	L 16# FFFF
KM	L KM 1111111111111111	2#	L 2# 11111111_11111111
KY	L KY 10,12	B#	L B# (10,12)
KT	L KT 10.0	S5TIME# (S5T#)	L S5TIME# 100ms
KC	L KC 30	C#	L C#30
DH	L DH FFFF FFFF	16#	L DW#16# FFFF_FFFF
KS	L KS WW	' xx '	L ' WW '
KG	L KG +234 +09	Floating point	L +2.34 E+08
Representation: S5 format ← Exponent → ← Mantissa → 31 30 24 23 22 0 SE 2^6.. ... ... 2^0 SM 2^{-1}..... .....2^{-23} Exponent = value of exponent SE = sign of the exponent SM = sign of the mantissa Range of values: 1.5 x 10^{-39} to 1.7 x 10^{38}		**Repr.: Single format compl. with ANSI/IEEE** ← Exponent → ← Mantissa → 31 30 23 22 0 S 2^7.. ... ... 2^0 2^{-1}.. .... 2^{-23} Exponent = actual exponent + bias* (+127) S = sign of the mantissa Range of values: approx. 1.18 x 10^{-38} to 3.4 x 10^{+38}	

* Bias: This is an offset factor separating the exponents into positive and negative areas. The value 127 in the exponent area corresponds to the value 0 in an absolute sense.

For further information about data types see the *Statement List Programming Manual* /**232**/.

3.11 Address Areas

3.11.1 Overview

Table 3-11 Addresses in S5 and S7

Address Areas	Addresses in S5	Addresses in S7	Remark
Inputs	I	I	
Outputs	Q	Q	
I/O	P, Q, G	PI for load commands	Shared I/O is not converted
		PQ for transfer commands	
Bit memory (flag) area	F	M	
	S	M	from M 256.0 (Converter)
	“Scratchpad flags”	L	Converted like flags
Timers	T	T	
Counters	C	C	
Data area	D...	DB...	Converted as shared data addresses
System data	RS, RT, RI, RJ	-	Not converted
Page area	C	-	

Note on Data Addresses

In S7 there are two data block registers: the DB register, which is predominantly used for shared data blocks and the DI register, which is preferred for instance DBs. For this reason there are also two types of data addresses. The addresses DBX, DBB, DBW, and DBD are addresses of shared data blocks; the addresses DIX, DIB, DIW, and DID are addresses of instance DBs. During conversion, addresses of shared data blocks are used for the data block addresses D, DB, DW, DD.

Also note how data blocks are converted (see Section 3.7.6).

Warning

Be aware that the size and number areas for address areas and the number and length of blocks for S7 all depend upon the CPU used. CPU performance criteria and ratings can be found in Section 2.2.1.

3.11.2 New Addresses in S7: Local Data

Local Data in STEP 7

Local data in STEP 7 are the data assigned to a logic block which are either declared in its declaration section or in its variable declaration. Depending on the block, they consist of formal parameters, static data, and temporary data. Local data are usually addressed symbolically.

Block Parameters

Block parameters of functions (FC) are handled like the block parameters in S5: the block parameters represent pointers which point to the corresponding actual parameter.

Block parameters of function blocks (FB) are stored like the static local data in the instance data block.

Static Local Data

Static local data can be used in every function block. They are defined in the declaration section and stored in the instance data block.

Static local data, like data addresses in shared data blocks, retain their value until they are overwritten by the program.

Generally, the static local data are only processed in the function block. However, since they are stored in a data block, the local data can be accessed in the user program at any time, as is the case with variables in a shared data block.

Temporary Local Data

Scratchpad flags in STEP 5

In STEP 5, bit memory address areas are used to store data temporarily within blocks. By common agreement, the flags 200 to 250 are reserved for temporary storage. The management of scratchpad flags is completely up to the user.

Temporary local data in STEP 7

Temporary local data are storage areas for data that are only valid during block processing. As soon as the block has been processed, these local data release the used memory again. Each priority class has its own local data stack. This prevents intermediate results from being inadvertently overwritten by interrupt programs.

Using Temporary Local Data in STEP 7

In STEP 7, temporary variables are used for the following three applications:

- As intermediate storage for data from a user program.

  This application is explained above and applies to functions (FCs), function blocks (FBs), and organization blocks (OBs).

- As memory used for transferring operating system information to the user program.

  The information supplied by the operating system to the user program has a special name: “start information.” This start information is exclusively provided to the organization blocks (OBs) as an interface between the operating system and the user program.

- To transfer parameters in FCs.

Where Are Temporary Local Data Declared?

You declare temporary local data within a block. When you create a new block, you declare symbolic names for the temporary variables at the beginning and then use them within the block. Each priority class has 256 bytes available in the S7-300. A total of 16 Kbytes are available in the S7-400 which the user can divide among the priority classes when assigning parameters to the CPU.

3.12 Instructions

The following table provides an overview of the instructions used. In addition, it also shows which instructions can be converted. If the instructions are not convertible, then other conversion options are indicated.

Table 3-12 Instructions in S5 and in S7

Instruction Type	S5	S7	Conversion	Conversion Option
Accumulator instructions	TAK, ENT, I, D, ADDBN, ADDKF, ADDDH	TAK, ENT, INC, DEC, +, New in S7: CAW, CAD, PUSH, POP, LEAVE	yes	–
Address register instructions / Register instructions	MA1, MBR, ABR, MAS, MAB, MSB, MSA, MBA, MBS; TSG, LRB, LRW, LRD, TRB, TRW, TRD	New in S7: LAR1, LAR2, TAR1, TAR2, +AR1, +AR2, CAR	no	Use address register (AR1, AR2)
Bit logic instructions	A, AN, O, ON, A(, O(,), O, S, R, RB, RD= TB, TBN, SU, RU	A, AN, O, ON, A(, O(,), O, S, R, = SET; A, SET; AN, SET; S, SET; R New in S7: X, XN, X(, XN(, FP, FN, NOT, SET, CLR, SAVE	yes	–
Timer instructions	SP, SE, SD, SS/SSU, SF/SFD, FR, SEC	SP, SE, SD, SS, SF, FR, S T	yes	–
Counter instructions	CU/SSU, CD/SFD, FR, SEC	CU, CD, FR, S C	yes	–
Load and transfer instructions	L, LD, LW, LDW, TL PB, L QB, L PW, L QW, T PB, T QB, T PW, T QW	L, LC, T L PIB, L PIW, T PQB, T PQW	yes	-
	LY GB / GW / GD / CB / CW / CD, LW GW / GD / CW / CD, TY GB / GW / GD / CB / CW / CD, TW GW / GD / CW / CD		no	Substitute by access to I/O area

(continued on next page)

Table 3-12 Instructions in S5 and in S7, continued

Instruction Type	S5	S7	Conversion	Conversion Option
Integer math instructions	+F, –F, xF, :F, +D, –D	+I, –I, *I, /I, +D, –D, *D, /D New in S7: MOD	yes	–
Floating-point math instructions	+G, –G, xG, :G	+R, –R, *R, /R	yes	–
Comparison instructions	!=F, ><F, >F, <F, >=F, <=F, !=D, ><D, D, <D, >=D, <=D, !=G, ><G, >G, <G, >=G, <=G	==I, <>I, >I, <I; >=I, <=I, ==D, <>D, >D, <D, >=D, <=D, ==R, <>R, >R, <R, >=R, <=R	yes	–
Conversion instructions	CFW, CSW, CSD DEF, DED, DUF, DUD, GFD, FDG	INVI, NEGI, NEGD, BTI, BTD, DTB, ITB, RND, DTR New in S7: ITD, RND+, RND–, TRUNC, INVD, NEGR	yes	–
Word logic instructions	AW, OW, XOW	AW, OW, XOW New in S7: AD, OD, XOD	yes	–
Shift and rotate instructions	SLW, SLD, SRW, SRD, SVW, SVD, RLD, RRD	SLW, SLD, SRW, SRD, SSI, SSD, RLD, RRD New in S7: RLDA, RRDA	yes	–
Data block instructions	G, CX	OPN	yes	
	G, GX	SFC22	no	Substitute by calling SFC22 CREATE_DB
		New in S7: CDB L DBLG, L DBNO, L DILG, L DINO		

(continued on next page)

Table 3-12 Instructions in S5 and in S7, continued

Instruction Type	S5	S7	Conversion	Conversion Option
Logic control instructions, jump	JU, JC, JN, JZ, JP, JM, JO, JOS, JUR	JU, JC, JN, JZ, JP, JM, JO, JOS New in S7: JCN, JCB, JNB, JBI, JNBI, JMZ, JPZ, JUO, LOOP, JL	yes	–
Block instructions	JU, JC, DOU, DOC, BE, BEU, BEC	CALL, BE, BEU, BEC	yes	–
Command output instructions/ Master control relay instructions	BAS, BAF	New in S7: MCRA, MCRD, MCR(,)MCR	no	Substitute by calling SFC26, SFC27 or master control relay instructions
Stop commands	STP, STS, STW	SFC46	no	Substitute by calling SFC46 STP
Processing functions	DO <Formal parameter>	-	no	Call of DB / code block must be newly programmed
	DO FW, DO DW	Memory-indirect addressing	yes	Recommendation: substitute with register indirect addressing
	DO RS	Area-crossing register-indirect addressing	no	Must be substituted with indirect addressing (see Section 3.13.4)
Absolute memory addressing	LIR, TIR, LDI, TDI	–	no	Must be substituted with indirect addressing (see Section 3.13.4)
Block transfers	TNB, TNW, TXB, TXW	SFC20	no	Substitute by calling SFC20 BLKMOV
Interrupt commands *(continued on next page)*	LIM, SIM, IAE, RAE, IA, RA	SFC39 to 42	no	Substitute by calling SFC39 - 42

Table 3-12 Instructions in S5 and in S7, continued

Instruction Type	S5	S7	Conversion	Conversion Option
Page commands	ACR, TSC, TSG	–	no	S7 has no page access.
Math functions	–	ABS, COS, SIN, TAN, ACOS, ASIN, ATAN, EXP, LN	–	–
Null instructions	BLD xxx NOP 0, NOP 1	BLD xxx NOP 0, NOP 1	yes	–

3.13 Addressing

3.13.1 Absolute Addressing

The absolute addressing in S5 and S7 is identical, with one exception:

In S7, data in data blocks are addressed **in bytes**; that is, word addresses in S5 are transformed into byte addresses by being multiplied by 2.

The following table shows the assignment during this conversion (data area addressing:

S5	S7
DL 0, 1, 2, 3, ...255	DBB 0, 2, 4, 6, ...510
DR 0, 1, 2, 3, ...255	DBB 1, 3, 5, 7, ...511
DW 0, 1, 2, 3, ...255	DBW 0, 2, 4, 6, ...510
DD 0, 1, 2, 3, ...254	DBD 0, 2, 4, 6, ...508
D x.y	DBX 2 x.y for $8 \leq y \leq 15$ DBX (2 x+1).y for $0 \leq y \leq 7$

3.13.2 Symbolic Addressing

The symbolic addressing in S5 is also used in S7. However, there are now new options for creating and using the symbols. There are no differences in programming.

Symbols in STEP 5

Symbols for STEP 5 programs are declared with the help of the symbol editor. This editor generates an assignment list which allows you to use the symbols defined there instead of absolute addresses.

Symbols in STEP 7

In S7, symbols can be up to 24 characters long.

Shared Symbols

STEP 7 also has a symbol editor, but the assignment list (ZULI) is now known as a “symbol table.” In it you can declare all shared symbols such as inputs, outputs, bit memory (flags), and blocks.

When you assign symbols with the symbol editor, these are valid for a CPU program.

Local Symbols

Besides being able to declare symbols with the symbol editor, STEP 7 also gives you the option of specifying local symbols for data addresses and for the local data area when programming blocks.

If you assign symbols within a block instead of assigning them with the symbol editor, then this symbol is only “valid” for the block concerned. In this case the symbol is “local to the block.”

When are Symbols Declared?

STEP 7 does not stipulate exactly when you have to specify your symbols. When doing this you have the following two options:

- Specify them before beginning to program

  (This is required if the user program is input incrementally; that is, if the program syntax is checked after each line is created.)

- Specify them after creating the user program but before compiling

  (This is required if the user program is input in free-edit mode; that is, if the program is created as an ASCII file (source file).)

Importing a Symbol Table

In S7, you have the option of creating and editing the symbol table with the editor of your choice.

You can import tables that you created with another tool into your symbol table and then edit them further. For example, the import function can be used to add assignment lists created in STEP 5/ST after conversion.

The following data formats are available: *.SDF, *.ASC, *.DIF, and *.SEQ.

To import a symbol table, proceed as follows:

1. Open the S7 program containing the symbol table in the project window.
2. Double-click the “Symbols” container to open the symbol table.
3. Select the menu command **Symbol Table ▸ Import** in the window containing the symbol table. A dialog box is displayed.
4. Select the symbol table that you want to import in the dialog box and then click the “Open” button.
5. Check over the data records in the symbol table and make any corrections necessary.
6. Save and close the symbol table.

Note

A symbol table in *.SEQ file format that was converted from S5 to S7 can no longer be imported into S5. The file format *.DIF is recommended for exchanging symbol tables between S5 and S7.

For further information on symbol tables, see the *User Manual* **/231/**.

3.13.3 New Feature: Complete Addressing of Data Addresses

Complete addressing means that the data block is specified along with the data address. This was not possible in S5.

Complete addressing can occur either absolutely or symbolically. Combining absolute and symbolic addressing within one statement is not possible.

Example

L DB100.DBW6

L DB_MOTOR.SPEED

DB_MOTOR is the symbol for the data block DB100 and is defined in the symbol table. MOTOR.SPEED is a data address that was declared in the data block. This means that the symbolic name for the data address (DB_MOTOR.SPEED) is just as unique as the absolute address (DB100.DBW6).

Completely addressed data access can only be done in connection with the shared data block register (DB register). During complete addressing the STL editor issues statements:

1. Open the data block via the DB register (such as OPN DB100)
2. Access the data address (such as L DBW 6)

Possible Operations Using Completely-Addressed Data Access

You have the option of using completely addressed access for all instructions that are allowed for the data type in the data address being addressed.

Completely addressed data addresses can also be specified as block parameters. This is strongly recommended since it is possible for the data block to be switched when it is called. Complete addressing ensures that the correct data address is transferred from the correct data block.

Dangers of "Partial Addressing"

In principle it is possible to access data addresses in the same way as in STEP 5 ("partial addressing").

Example:

```
L DBW 6
L SPEED
```

In STEP 7 this may cause problems because STEP 7 changes the registers for the S7-300/S7-400 CPU during various operations. In some cases the DB number in the DB register will be overwritten.

The DB register may be overwritten in the following situations. Thus, particular care must be taken here:

- The DB register is overwritten during data access using complete addressing.
- If a function block (FB) is called, then the data block register for the calling block is overwritten.
- After a call is made to a function (FC) which transfers a parameter with a complex data type (such as STRING, DATE_AND_TIME, ARRAY, STRUCT, or UDT), the contents of the DB register for the calling block are overwritten.
- After you have assigned an actual parameter to an FC stored in a DB (such as DB100.DBX0.1), STEP 7 opens the DB (DB100) in which the contents of the DB register are overwritten.
- After an FB has addressed an in/out parameter with a complex data type such as STRING, DATE_AND_TIME, ARRAY, STRUCT, or UDT, STEP 7 uses the DB register to access data. This step overwrites the contents of the DB register.
- After an FC has addressed a parameter (input, output or in/out) with a complex data type (such as STRING, DATE_AND_TIME, ARRAY, STRUCT, or UDT), STEP 7 uses the DB register to access data. This step overwrites the contents of the DB register.

3.13.4 Indirect Addressing

Indirect addressing using the “DO” function of S5 has been replaced in S7 by the new indirect memory and register addressing commands.

Pointer Format in STEP 5

In S5 the pointer for the indicated processing operation occupies one word. The structure of the pointer is depicted in Figure 3-7:

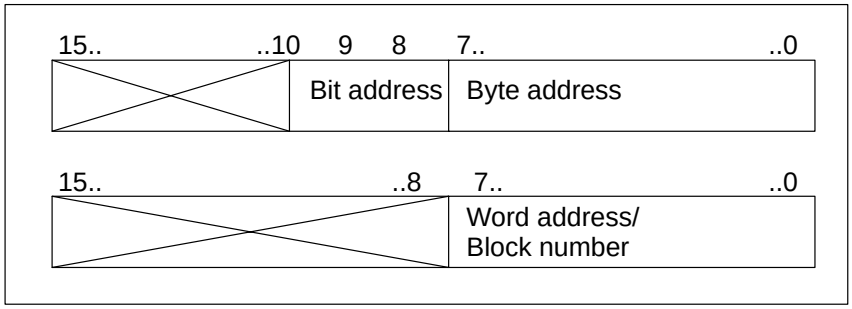

Figure 3-7 Structure Pointer S5

Pointer Format in STEP 7

In S7 there are two possible pointer formats, word and double-word.

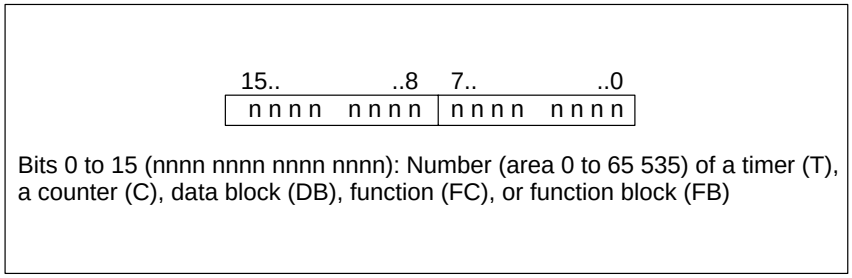
15..
..8
7..
..0
n n n n
n n n n
n n n n
n n n n
Bits 0 to 15 (nnnn nnnn nnnn nnnn): Number (area 0 to 65 535) of a timer (T), a counter (C), data block (DB), function (FC), or function block (FB)

Figure 3-8 Pointer in Word Format for Memory-Indirect Addressing

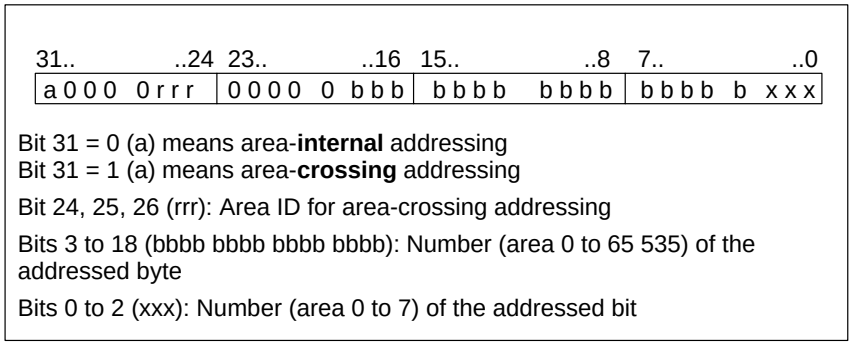
31..
..24
23..
..16
15..
..8
7..
..0
a 0 0 0
0 r r r
0 0 0 0
0
b b b
b b b b
b b b b
b b b b
b
x x x
Bit 31 = 0 (a) means area-internal addressing
Bit 31 = 1 (a) means area-crossing addressing
Bit 24, 25, 26 (rrr): Area ID for area-crossing addressing
Bits 3 to 18 (bbbb bbbb bbbb bbbb): Number (area 0 to 65 535) of the addressed byte
Bits 0 to 2 (xxx): Number (area 0 to 7) of the addressed bit

Figure 3-9 Pointer in Double Word Format for Memory-Indirect and Register Indirect Addressing

Memory-Indirect Addressing

Memory-indirect addressing corresponds to indirect addressing in S5. During memory-indirect addressing, the address specifies the address of the value that will process the instruction. The address consists of the following parts:

- An address identifier, such as “IB” for “input byte”, and
- A word that contains the number of a timer (T), a counter (C), a data block (DB), a function (FC) or a function block (FB), or
- A double word that specifies the exact address of a value within the memory area indicated by the address identifier.

The address uses the pointer to indirectly indicate the address of the value or the number. This word or double word can be located in one of the following areas:

- Bit memory (flag) (M)
- Data block (DB)
- Instance data block (DI)
- Local data (L)

The advantage of memory-indirect addressing is that you can dynamically modify the address of the statement when editing the program.

Examples

The following examples show how you can work with a pointer in word format:

STL S5	STL S7	Explanation

`L KB 5`
`T FW 2`
`DO FW 2`
`L T 0`
	`L +5`
	`T MW 2`
	
	`L T [MW 2]]`
		Load the value 5 as an integer in ACCU 1.
		Transfer the contents of ACCU 1 into the memory word MW2.
		Load the time value of the timer T 5.

The following two examples show how you can work with a pointer in double-word format.

STL S5	STL S7	Explanation

`L KB 8`
`T FY 3`
`L KB 7`
`T FY 2`
`DO FW 2`
`A I 0.0`
`DO FW 2`
`= Q 0.0`
	`L P#8.7`
	`T MD 2`
	
	`A I [MD 2]`
	
	`= Q [MD 2]`
		Load 2#0000 0000 0000 0000 0000 0000 0100 0111 (binary value) in ACCU 1 (S7).
		Save the address 8.7 in the memory word FW 2 (S5) / memory double word MD 2 (S7).
		
		The controller queries the input I 8.7 and assigns its signal state to the output Q 8.7.

STL S5	STL S7	Explanation
L KB 8 DO FW 2 DO FW 2 L IB 0 DO FW 2 T FW 0	L P#8.0 T MD2 L IB [MD2] T MW [MD2]	Load 2#0000 0000 0000 0000 0000 0000 0100 0000 (binary value) in ACCU 1 (S7). Save the address 8 in memory word FW 2 (S5) / memory double word MD 2 (S7). The controller loads input byte IB 8 and transfers the contents to memory word FW 8 (MW 8 in STEP 7).

Using the Correct Sequence Syntax

When working with a memory-indirect address that is stored in the memory area of the data block, you must first open the data block by using the "Open data block" instruction (OPN). After this, you can use the data word or data double word as the indirect address, as shown in the following example:

```
OPN     DB10
L       IB [DBD 20]
```

When accessing a byte, word, or double word, first make sure that the bit number of the pointer is "0."

Register-Indirect Addressing

In STEP 7, the address registers AR1 and AR2 are used for indirect addressing.

With indirect addressing, the address specifies the memory location of the value that will process the instruction. The address consists of the following two parts:

- An address identifier
- An address register and a pointer for indicating the offset added to the content of the address register in order to determine the exact address that the instruction is to process. The pointer is indicated by P#Byte.Bit.

The address points indirectly to the address of the value. It does this by using the address register plus the offset.

An instruction that uses area-internal, register-indirect addressing does not change the value in the address register.

For further information, see the *Statement List Programming Manual* /**232**/.

Part 2: Converting Programs

Procedure 4

Preparing for Conversion 5

Conversion 6

Editing the Converted Program 7

Compiling 8

Application Example 9

Procedure

4

The programming of S7 in STL is largely compatible with S5 STL. Similarly, programming Ladder in S7 is compatible to S5 LAD and programming FBD in S7 is compatible to S5 CSF. Thus, if you are an S5 user and want to use existing programs in S7, this change is made much easier for you. You can base the new system on your tried and tested S5 programs and convert them to S7 programs.

How to Proceed

The following list tells you how to convert your S5 program and lists the sections where you will find the required information.

The list is intended as an example and as a guideline (individual steps can also be skipped).

4.1 Analyzing the S5 System

Before you convert your S5 program you should clarify the following questions:

Functionality of the Modules (see Chapter 2)

How can the functionality of the modules used in your S5 program be achieved in S7? Can your S5 modules be used in S7 with the help of adapter or interface modules? Can your S5 modules be replaced with S7 modules?

System Settings (see Section 3.8)

How can the required system settings be implemented in S7?

Range of Instructions (see Section 3.12)

How can the range of instructions used by the S5 CPU be implemented using your S7 CPU?

If individual instructions cannot be converted, a message is output indicating the corresponding program sections and the instructions must be reprogrammed manually.

Standard Software (see Section 3.9)

Do the S5 standard function blocks called in the program to be converted also exist as functions in S7?

The S7 Standard software supplied includes the standard software packages already converted for basic functions, floating-point math, integrated functions, signal functions, and math functions.

Special Functions (see Table 3-5)

Can integrated special functions used in S5 programs be replaced?

Which Parts of Your Program Should be Reprogrammed in S7?

In general, not all the parts of a program can be converted. Considering the following points will help you decide whether to convert your S5 program or to recreate it in S7.

- Programs only containing digital and binary logic operations do not need to be revised.
- Absolute addresses cannot be addressed in S7. The corresponding instructions (such as LIR, TIR, etc.) are not converted. If a program frequently works with absolute addresses, it is a good idea to rewrite these parts of the program or, if necessary, the entire program.
- Processing functions (such as DO FW, DO DW) are partially converted; however, in this case you can save memory by reprogramming the functions in S7. This functionality can be obtained with indirect addressing.
- The parameter values of block calls must be always be checked and adapted since the actual parameters used are transferred during conversion without being changed.

4.2 Creating an S7 Project

STEP 7 provides you with the following two options for creating a project:

Creating a Project with the STEP 7 Wizard

The STEP 7 wizard helps you create a STEP 7 program quickly with the CPU you want to use. After completing this step, you can start programming.

Creating a Program Manually

In addition, you have the option of creating your project manually. The procedure for this is described in Section 3.3.1.

4.3 Defining Hardware

At this point it is a good idea to configure the hardware since data are determined in HWConfig that can be then be used to prepare for conversion.

However, if you do not want to configure your hardware yet, you can still do it later.

Defining Hardware

The information found in Chapter 2 (“Hardware”) will help you select the S7 or S5 modules required for your configuration and fill out the hardware configuration table (see Section 3.4).

Address Allocation

The address allocation for the modules is done automatically by HWConfig. This means that you can already use the addresses during conversion.

Making System Settings

When assigning parameters to the CPU in HWConfig you can also make system settings which were created in DB1/DX0 in S5 or by system utilities (see Section 3.4).

Specifying Retentive Behavior

The retentive behavior can also be set in the parameter data in the CPU. The retentive behavior is, however, dependent on the battery backup (see Section 3.4).

Preparing for Conversion

5

Overview

Providing the required files (see Section 5.1)	• Program file <Name>ST.S5D • Cross-reference list <Name>XR.INI • Optional assignment list <Name>Z0.SEQ
Checking addresses (see Section 5.2)	• Number of addresses • Number of blocks
Preparing the S5 program (see Section 5.3)	• Evaluate and delete the data blocks DB1 / DX0 • Remove calls from the integrated blocks • Remove access to the system data area • Adapt the address areas • Assign macros to non-convertible program parts • Delete data blocks without structure down to one data word
Creating macros (see Section 5.4)	• Command macros • Organization block (OB) macros

5.1 Providing the Required Files

The following data are required as the basis for converting your S5 program:

- Program file <Name> ST.S5D and
- Cross-reference list <Name> XR.INI

The cross-reference list is required when converting in order to retain the program structure and call hierarchy of the S5 program.

Optional Requirement

If you want to use symbolic names in your program instead of absolute addresses, you also require the following data to generate the converted assignment list:

- S5 assignment list <Name> Z0.SEQ.

Procedure

To prepare the conversion, proceed as follows:

1. Create a current cross-reference list for your S5 program using the S5 software.
2. Copy your STEP 5 program file, the corresponding cross-reference list and, if necessary, the assignment list into a DOS directory.

5.2 Checking Addresses

Range of Functions of the CPU

It may be necessary to adapt the converted program to the S7 CPU being used.

To gain an overview of the range of functions of the S7 CPU, proceed as follows:

1. Determine the S7 CPU that you want to use.
2. Find this S7 CPU in the performance specifications tables in Section 2.2.1 and compare the following two specifications:

- Number of addresses
- Number of blocks

with the addresses and blocks to be used,

or

1. Open the SIMATIC Manager.
2. Select the S7 CPU in the online view of the project structure.
3. Use the menu command **PLC ► Module Information** to open a dialog box which includes, among other things, the following information:

- In the “General” tab you can identify the CPU type, obtain information on the memory configuration, and read the size of the available address areas.
- In the “Blocks” tab there is information on the available blocks. This includes the maximum number and length of the blocks types, as well as all OBs, SFBs, and SFCs present on the CPU.

Adapting the Converted Program

To adapt the STL program being converted so that it can run on the CPU, check it for the permitted number of blocks and addresses, and modify as necessary.

5.3 Preparing the S5 Program

Before actually converting your STEP 5 program, you can prepare it for its future use as a STEP 7 program. (However, you do not have to do this first; all necessary corrections can also be made in the STEP 7 source file after the conversion.) This initial adaptation will reduce the number of error messages and warnings occurring during conversion.

For example, you can make the following adaptations to the STEP 5 program before proceeding with the conversion:

- Evaluate system settings in the data blocks with the program properties DB1 or DX0. After this, you can delete DB1 and DX0.
- Remove all calls from integrated blocks or accesses to the system data area; this functionality can be achieved by assigning parameters to the S7 CPU.
- Adapt all input, output, and peripheral address areas to the (new) module addresses by using the STEP 5 function “Rewire.” (When doing this, you should make sure that the STEP 5 address area is not exceeded; otherwise, an error will be reported during the first cycle of the conversion process. If this occurs, these instructions will not be converted.)
- Delete all repeated non-convertible parts of the program until there is only one “unique” STEP 5 instruction for each part of the program. This “unique” instruction can be assigned a macro to replace the part of the program (see Section 5.4)
- If your program contains very many (and long) data blocks having no structure (such as those used as data buffers), you can delete the data words in these data blocks until only one data word remains. After converting but before compiling, you can program the contents of these data blocks in the source file by using an array declaration, such as buffer: ARRAY [1..256] of WORD.

With the converter you can not only convert complete programs but also individual program blocks.

5.4 Creating Macros

Uses of Macros

When converting, you can define macros for the following:

- S5 instructions that cannot be automatically converted and
- S5 instructions that you want to convert differently from the standard conversion.

Macros can be useful if your program contains many S5 instructions which correspond to the characteristics listed above.

Macro Functions

Macros can replace the following:

- S5 instructions
- S5 organization blocks (OBs)

The macros are saved for the SIMATIC instruction set in the S7S5CAPA.MAC file and for the international instruction set in the S7S5CAPB.MAC file. If you work with both instruction sets, you must specify the macros for each file. A distinction is made between instruction macros and OB macros. You can create 256 instruction macros and 256 OB macros.

5.4.1 Instruction Macros

Instruction macros must be structured as follows:

```
$MACRO: <S5 instruction>
S7 instruction sequence
$ENDMACRO
```

When defining a macro, enter the complete statement (instruction and absolute address) for <S5 instruction>.

The table below shows a macro for the statement G DB 0, which is used to set up data blocks in S5. The length (in words) of the data block to be set up is in ACCU 1. In S7, the function is realized using the system function SFC22 CREAT_DB. The length of the data block must be converted into bytes.

Table 5-1 Example of an Instruction Macro

Macro	S5	S7

`$MACRO: G DB 0 //Replaces instruction`
`//for setting up a DB`
 	
`SLW 1 //Number of words`
`//into number of bytes`
`T MW 102`
`CALL SFC 22( //Call SFC CREAT_DB`
`LOW_LIMIT := MW 100,`
`UP_LIMIT := MW 100,`
`COUNT := MW 102,`
`RET_VAL := MW 106,`
`DB_NUMBER := MW 104);`
`$ENDMACRO`
	`L Constant`
	`DO FW 100`
	
	`G DB 0`
		`L Constant;`
		
		`SLW 1;`
		`T MW 102;`
		`CALL SFC22(`
		`LOW_LIMIT := MW 100,`
		`UP_LIMIT := MW 100,`
		`COUNT := MW 102,`
		`RET_VAL := MW 106,`
		`DB_NUMBER := MW 104);`

5.4.2 OB Macros

Due to the differences in the organization blocks between S5 and S7 it may be advisable to control the conversion of your instructions with S5 OBs yourself. In this case, OB macros must be structured as follows:

```
$OBCALL: <Number of the OB>

CALL <S7 system function>;

$ENDMACRO
```

If an instruction with the address OB x is found in the S5 source file, this instruction is replaced by the defined macro instructions. Exceptions to this are the FB calls that use OBs as formal parameters.

Table 5-2 Example of an OB Macro

Macro	S5	S7

`$OBCALL: 31 //Replaces instructions //with OB31`
`CALL SFC 43;`
`$ENDMACRO`
	`JU OB 31`	`CALL SFC43;`

Notes on Creating OBs

The functions of the organization blocks in S5 are different from those of the OBs in S7. OBs that cannot be converted automatically must be replaced by the following:

- OBs with a different range of functions
- New S7 instructions, or
- System settings which are defined when assigning the hardware parameters

For detailed information about replacing S5 OBs, see Section 3.7.5.

Note

There is no check to determine whether a macro is defined twice. If this happens to be the case, then the first macro defined is used. There is no check to determine whether the specified S7 instruction sequence is correct. Make sure that keywords and special characters (colon) are correctly written.

5.4.3 Editing Macros

Macros are created as follows:

1. Start the S5/S7 Converter by clicking the “Start” button in the Windows 95 taskbar and selecting Simatic/STEP 7/Convert S5 files.
2. Select the menu command **Edit ▸ Replace Macro** (There must be no program file open!).

Result: The S7S5CAPB.MAC file is opened.

3. Enter the macros as described above and save the file with the menu command **File ▸ Save**.
4. Close the file with the menu command **File ▸ Close**.

Result: The S7S5CAPB.MAC file is closed. The macros are valid the next time you start a conversion run.

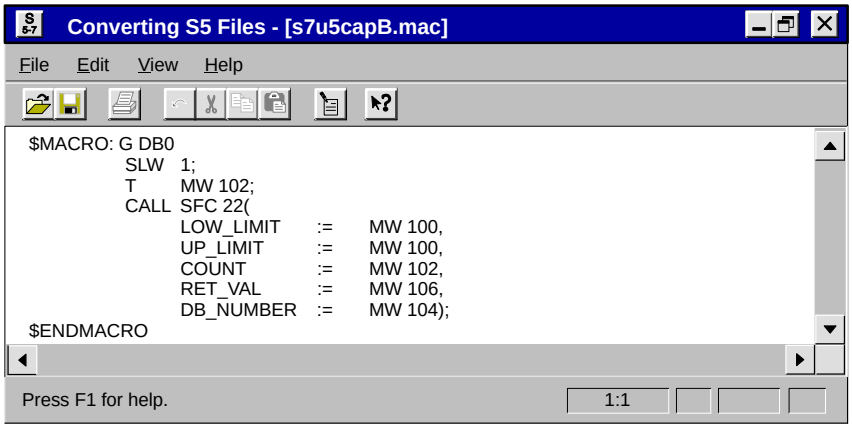

Figure 5-1 Macro in the Window “Converting S5 Files”

Conversion

6

6.1 Starting the Conversion

Prior Requirements

Before you start to convert programs, make sure that the S5 file you want to convert, the cross-reference list and, if necessary, the assignment list are in the same directory (see Section 5.1).

Starting the S5/S7 Converter

After you have installed the STEP 7 software on your programming device, start the S5/S7 Converter using the “Start” button in the taskbar of Windows 95.

- Click on the entry “Simatic/STEP 7/Convert S5 files”.

The S5/S7 Converter then displays the following initial screen:

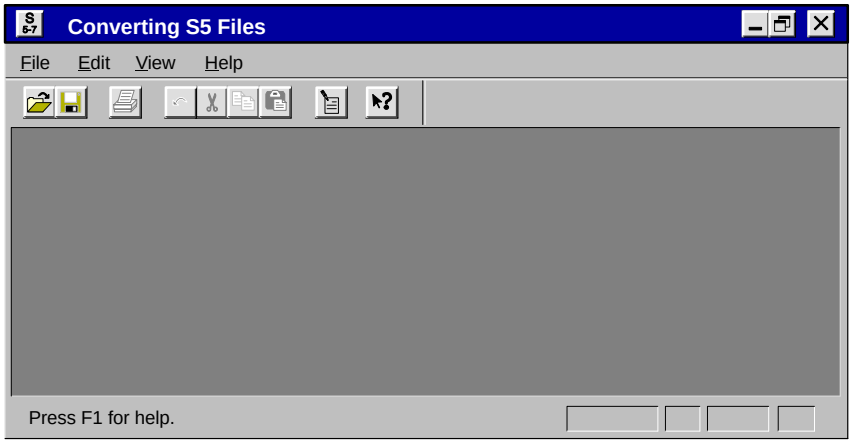

Figure 6-1 Initial Screen of the S5/S7 Converter

Selecting a Program File

To select a program file, proceed as follows:

1. Select the menu command **File ▸ Open**.
2. Select the drive and the directory containing the files to be converted.
3. Select the file to be converted and click “OK” to confirm your selection.

Result: The S5/S7 Converter displays the source and target files and an assignment of the old and new block numbers.

The figure below shows the dialog box “Converting S5 Files [<Test>ST.S5D]”.

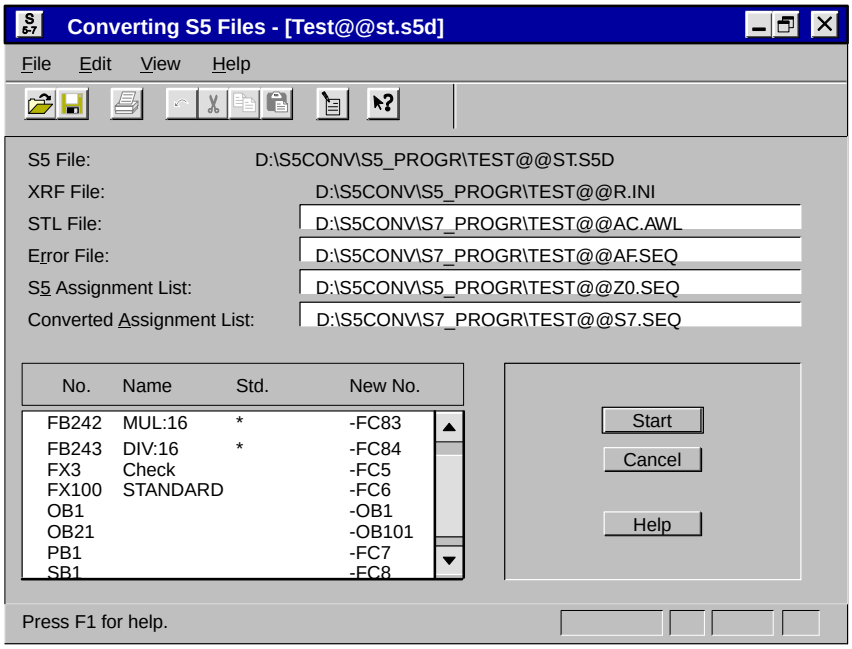

Figure 6-2 “Converting S5 Files – [<Test>ST.S5D]” Dialog Box

Changing the Target File Names

If required, you can modify the names of the target files “STL File”, “Error File” and “Converted Assignment List” proposed by the software. This may be necessary if the editor with which you want to process the converted files requires certain name conventions (for example TEST.TXT).

To change the name of a file, proceed as follows:

1. Click the text box with the path name of the target file you want to modify.
2. Modify the text as required.

Assignment No. –> New No.

The software proposes new numbers for the blocks to be converted and displays them in the dialog box “Converting S5 Files [<Test>ST.S5D]”. If you want to assign different numbers, proceed as follows:

1. Double-click the block number you want to modify.
2. Enter the new number in the “New Block Number” dialog box and click the “OK” button to confirm your entry.

S5 Standard Function Blocks

If your S5 program contains standard function blocks (SFBs), these are marked by an asterisk in the “Std.” column.

Starting the Conversion

By clicking on the “Start” button, you start the conversion. The conversion consists of two conversion runs and the conversion of the assignment list.

In the first conversion run, the S5 program is converted into an S5 source file with all blocks and comments.

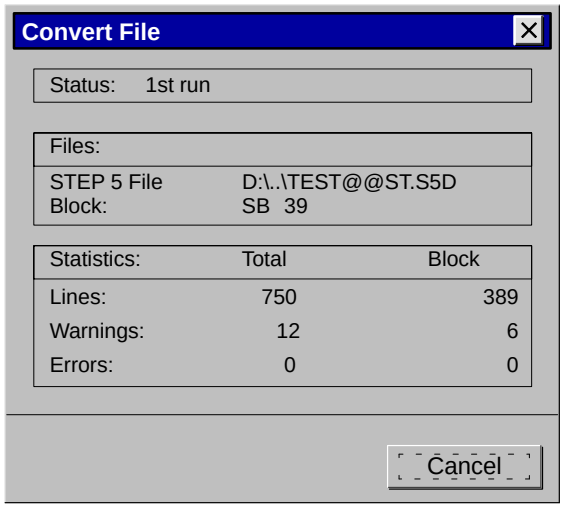

Figure 6-3 First Conversion Run

In the second run, the S5 source file is converted to the STL source file with the new block types, block numbers, and S7 syntax.

Converting the Assignment List

The symbols in the S5 assignment list are converted into a form which can be imported by the Symbol Editor.

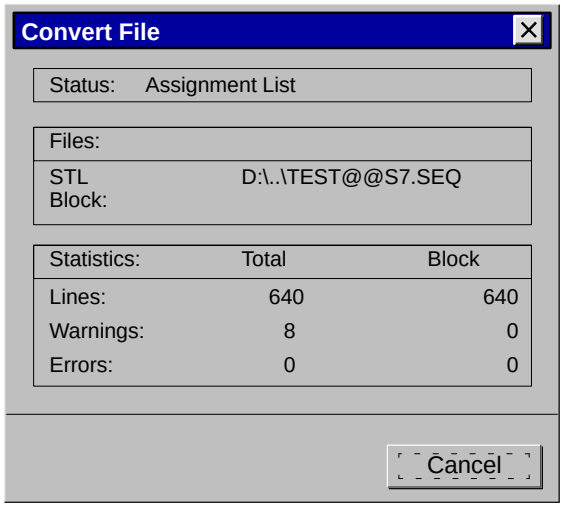

Figure 6-4 Converting the Assignment List

6.2 Generated Files

During conversion, the S5/S7 convertor generates the following files:

- The file <Name>A0.SEQ:

  This file is generated during the first conversion run. It contains the file <Name>ST.S5D in ASCII form.

- The file <Name>AC.AWL:

  This file is generated during the second conversion run. It contains the STL program. Any messages resulting from incorrect macro definitions originate from this run.

- The file <Name>S7.SEQ:

  This file is generated from the conversion of the assignment list. It contains the converted assignment list in a form suitable for importing with the Symbol Editor.

- The error file <Name>AF.SEQ:

  This file is displayed in the upper list box in the “Converting S5 Files” window and contains the errors and warnings in the converted program. These messages are generated during both conversion runs and also during conversion of the assignment list.

After the conversion runs are completed, a dialog box displays the total number of errors and warnings made in the converted program.

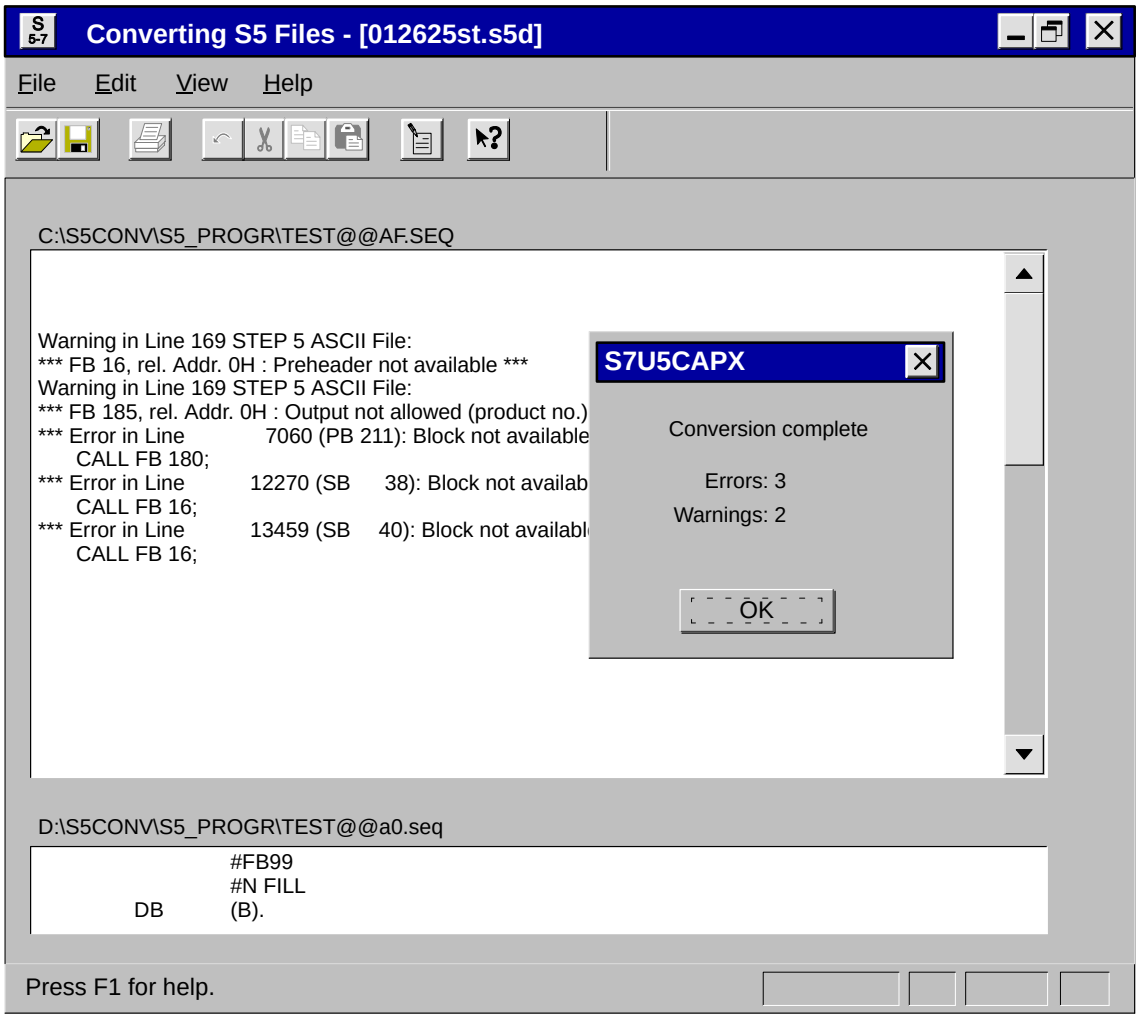
Converting S5 Files - [012625st.s5d]
File Edit View Help
C:\S5CONV\S5_PROGR\TEST@@AF.SEQ
Warning in Line 169 STEP 5 ASCII File:
*** FB 16, rel. Addr. 0H : Preheader not available ***
Warning in Line 169 STEP 5 ASCII File:
*** FB 185, rel. Addr. 0H : Output not allowed (product no.)
*** Error in Line 7060 (PB 211): Block not available
CALL FB 180;
*** Error in Line 12270 (SB 38): Block not availab
CALL FB 16;
*** Error in Line 13459 (SB 40): Block not availabl
CALL FB 16;
S7U5CAPX
Conversion complete
Errors: 3
Warnings: 2
OK
D:\S5CONV\S5_PROGR\TEST@@a0.seq
#FB99
#N FILL
DB (B).
Press F1 for help.

Figure 6-5 Messages When Converting

Localizing Errors

In the lower list box in the window you can display the location in the file at which the error occurred.

Messages are output in the STL source file at the points in the program at which errors were detected. This file also contains warnings or indications that problems might occur (for example, due to changes in the instruction semantics).

Printing Messages

Select the menu command **File ▸ Print** to print out the message files you require.

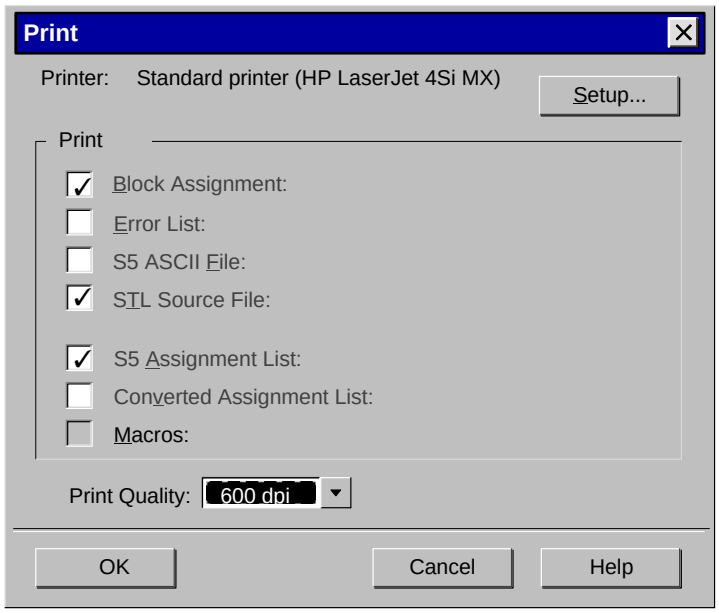

Figure 6-6 “Print” Dialog Box

6.3 Interpreting Messages

Analyzing Messages

The messages generated during conversion consist of error messages and warnings. To analyze these messages, proceed as follows:

1. Display the file containing the error in the lower list box of the “Messages” window.
2. The meaning of the messages can be found in the online help.
3. Correct the error as suggested under “Remedy.”

Error Messages

Error messages are displayed if parts of the S5 program cannot be converted and are only included as comments in the S7 program. The table below lists all the error messages, their meaning, and possible remedies.

References to Rules

Chapter 3 (Software) contains the rules for converting S5 programs to S7. Here you can also find further references to possible error sources and assistance when subsequently editing the STL program.

Table 6-1 Error Messages, Meaning, and Remedy

Error Message	Source	Meaning	Remedy
Absolute parameter does not match address	1st run	Wrong address ID	Check the instruction
Bit access to T/C is no longer allowed (please check)	2nd run	S5 program contains bit access to timers and counters	Check the STL program
Block not available	1st run	Called block (FB, FX) missing or block is shown in the block list but it does not exist in the program file	Check the program structure
	2nd run	Block is called that does not exist in the program file	Check whether the cross-reference list was specified, or check the program structure
CALL OB is not allowed	2nd run	Calling OBs is not allowed in S7	If necessary, use the statement CALL SFC
CALL SFC xy generating, please extend parameter list	2nd run	Parameters for SFC missing	Complete the SFC parameter list
Command in block not allowed	1st run	For example, jump within a program block	Check the instruction
Comment too long	1st run	Error in S5 file	Check the program file
Conversion error	2nd run	BI without constant	Include a constant with the load instruction
Directory not available	1st run	Program file does not contain any blocks	Check the program file
Error in macro file. Macro xy ignored	2nd run	Macro error	Check the macro instruction
Error in parameter	1st run	Error in the S5 program	Check the program file

Table 6-1 Error Messages, Meaning, and Remedy

Error Message	Source	Meaning	Remedy
File not found	general	Selected file does not exist	Check the program file
Invalid MC5 code was converted	1st run	Conversion of an older S5 instruction	None
Invalid operator	1st run	Operator in S5 file not known or cannot be converted	Replace the operator with the appropriate S7 instruction
Invalid operator, may be replaced by the instruction \"L P# formal parameter\"	2nd run	The operator cannot be loaded into S7 in this form	You may have to use the specified instruction
Jump label cannot be generated	2nd run	JUR instruction exceeds block limit	Correct the error in the S5 program
Label invalid	1st run	Jump label contains invalid characters	Check the S5 file
Label undefined	1st run	Jump label not defined in the preheader	Check the S5 file
Memory overflow in programming device (space problem)	1st run	Not enough main memory	Delete files you no longer require in the main memory
No access rights	general	File is read-only	Clear the read-only attribute
No block name given	1st run	Block name consists of only blanks	Enter a block name
Undefined command	1st run	Invalid MC5/STL instruction	Correct the S5 program file
	2nd run	Instruction does not exist in S7	Edit a macro or replace the instruction with the appropriate S7 instruction sequence
Undefined formal parameter	1st run	More parameters than in calling block	Check the S5 program file
Write error on diskette	general	File is read-only or there is no space on the diskette	Clear the read-only attribute or delete unnecessary data
Wrong address	1st run	Address does not match instruction	Check the S5 source file
	2nd run	Address does not match instruction	Modify the STL file
Wrong comment length	1st run	Error in S5 file	Check the program file
Wrong nesting depth	1st run	End of bracketed expression incorrect	Check the nesting levels, correct the programming error
Wrong number of parameters	1st run	Error in the S5 program	Check the program file
Wrong parameter type	1st run	Error in the S5 program	Check the program file

Warnings

Warnings are displayed if parts of the S5 program are converted but should be checked once more.

Table 6-2 Warnings, Meaning, and Remedy

Warning	Source	Meaning	Remedy
ID only influences Accu 1-L, now whole Accu 1	2nd run	S7 accumulators extended to 32 bits	Check the consequences of an indirect INCREMENT/DECREMENT instruction in the STL program
If S5 115U, then change to OB 100	2nd run	The startup OB21 in S5 is automatically converted to OB101	If the S5 program ran on an S5-115U, you have to change OB101 to OB100
Jump instruction after EDIT cannot be compiled	2nd run	An EDIT instruction with JU cannot be converted automatically	Replace the instruction in the STL file by JL and check the jump
Note block numbers may be changed	2nd run	An indirect block call does not take into account new block numbers (number is fetched from corresponding memory word or data word)	Change the logic in S5 or use fixed block calls
OB 23 and OB 24 have been converted to OB 122	2nd run	OB 23 and OB 24 are both replaced by OB 122 in S7	Put the contents of OBs 23 and 24 into OB 122 and delete the other OB 122
OB was interpreted as OB 34 from S5-115U	2nd run	Depending on the CPU used, the OB 34 can have different meanings	Check whether this OB matches your program
Output not allowed (GRAPH5 block)	1st run	GRAPH 5 blocks cannot be converted	You may have to insert a GRAPH 7 block
Output not allowed (product no.)	1st run	S5 standard function block must be replaced by an S7 FC	None
Please check time interval settings	2nd run	Time intervals can be more precisely set in S7 than in S5	Adjust the time interval using the function “Hardware Configuration”
Please observe different STOP commands	2nd run	No distinction has been made between STP, STS, and STW	Check the program file
Preheader not available	1st run	For FBs and FXs the jump label identifiers are missing, for DBs and DXs the data formats are missing	Check whether the preheaders exist in another file
RLO is set	2nd run	With the S5 instructions SU and RU the RLO is set in S7	If necessary, insert the instruction CLEAR
S5 screen DB was not used to assign parameters to S7	1st run	MASK is in DW0 and DW1	Assign parameters to the programmable controller using STEP 7
System preferences cannot be set by the S5/S7 Converter	2nd run	DB and DX will be converted but do not have the same meaning as in S5	Make the system settings using the configuration table

Editing the Converted Program

7

Preparing to Edit

The following preparations are necessary to edit the STL source file generated during conversion:

- Make a printout of the messages generated during conversion.
- Create an S7 program in a project in the SIMATIC Manager, if you have not already done so.
- Import the STL source file program into the “Source Files” container of this S7 program, using the menu command **Insert ▸ External Source File,**
- Open the converted file.

Editing the File

To edit the generated STL source file, we recommend the following procedure:

- Work through the program in interactive mode and modify or supplement the S5 instructions and organization blocks that were not converted based on the warnings (see Part 1).

7.1 Address Changes

Usually, input and output modules are affected by address changes. The addresses for these modules can be found in HWConfig.

7.1.1 Options for Changing Addresses

Rewiring in S5

Before converting you can use the “Rewire” function to adapt the addresses of individual addresses in S5 to the new addresses in S7.

Rewiring in S7

The SIMATIC Manager contains a function for automatically rewiring blocks generated from your source file.

To rewire blocks, proceed as follows:

1. Select the blocks in your program to be rewired in the SIMATIC Manager.
2. Open the table used for rewiring by selecting the menu command **Options ▸ Rewire**.
3. Enter the old and new addresses for each address in the table and then save them.

The blocks now contain the changed addresses.

Changing Addresses in the S7 Source File

In your program, adapt access to inputs and outputs as well as direct I/O access to the new module addresses in S7.

In the S7 source file you can easily make changes to the absolute addresses by selecting the menu command **Edit ▸ Replace**.

Caution: If the old and new address areas overlap, then unintended changes can occur.

Generating a New (Symbolically Addressed) S7 Source File

If you want to use symbolic addressing, you can also use the symbol table to do the rewiring.

Prerequisite

Before rewiring, you must already have a compiled program that is error-free and a symbol table that contains all the symbols necessary for modifying the absolute addresses.

Procedure

To change the addresses, proceed as follows:

1. Open a block containing addresses to be changed. Select the option "Symbolic Representation" in the "Editor" tab of the dialog box opened with the menu command **Options ▸ Customize**.
   Repeat this procedure for all blocks containing addresses that you wish to change.
2. Generate a source file from the blocks by selecting the menu command **File ▸ Generate Source File**. The blocks can be selected in a dialog box after you have entered the name of the source file.

When creating a sequence of blocks, remember to take the call hierarchy into account. As a rule, called blocks must already exist. This means that they must be entered in the source file in front of the blocks from which they are called.

Result: The source file generated contains the instructions with symbolic addressing.

3. Now you can carry out the rewiring in the symbol table. Replace the changed S5 addresses with the new S7 addresses.
4. Once the source file is compiled, the blocks contain the new addresses.

7.2 Non-Convertible Functions

Addresses and instructions that cannot be converted automatically are only included as comments in the generated S7 program. These you must revise yourself.

As the user, there are two ways in which you can convert these instructions:

- You can define your own S7 STL instruction sequence (macros) for these instructions (if they occur in the user program). These can then be used during conversion.
- You can edit the instruction sequences in the resulting S7 program.

Which method is better depends on the number of occurrences of such instructions in your user program.

You can read about non-convertible addresses and instructions in Sections 3.11 and 3.12. These sections also contain suggestions for creating non-convertible functions in S7.

7.3 Indirect Addressing – Conversion

The S5/S7 Converter uses STEP 7 instructions to convert indirect addressing with DO FW and DO DW. The instruction sequence generated is generally very extensive since the STEP 5 pointer has to be converted into STEP 7 format, and the accumulator contents and the status word must be buffered when doing so.

If your program contains very frequent indirect addressing, then it is worth adapting to the indirect addressing in STEP 7. A substantial amount of memory space can be saved by using appropriate programming techniques.

The list below explains how the S5/S7 Converter converts indirect addressing in different cases:

Timers and Counters

Indirect addressing of timers and counters is converted into memory-indirect addressing by using a temporary local data word.

Blocks

Indirect addressing of blocks is converted into memory-indirect addressing by using a temporary local data word.

The new block numbers cannot be taken into account during conversion and must therefore be corrected.

Addresses

The indirect addressing of addresses is converted by bits and words into register-indirect addressing by using the address register AR1 and temporary local data as a buffer for the status word STW, ACCU 1, and ACCU 2.

Indirect Addressing via the BR Register

The instructions are not converted. Indirect addressing must be reprogrammed in S7.

Other Types of Indirect Addressing

The instructions must be reprogrammed in S7.

For further information on indirect addressing, see Section 3.13.4.

7.4 Working with Direct Memory Access

In STEP 5, access to absolute memory addresses was used for some functions. This type of access no longer exists in STEP 7.

STEP 5	STEP 7
Addressing data addresses in “extra long” data blocks	Addressing data addresses greater than 255 can now be done with standard instructions (L, T, ...).
Indirect addressing with the BR register	Indirect addressing can be done with register-indirect addressing (see Section 3.13.4 and the *Statement List Programming Manual* **/232/**).
Using block transfers	For block transfers there is now a system function SFC20 BLKMOV. The memory areas to be copied are specified at the block parameters. If the memory areas are variable, then they can be specified at the parameters “ANY pointer”, which can be accessed in the user program.

7.5 Assigning Parameters

S5 Command B<Block Parameters>

Depending on the type of block transferred, the statement B <Formal Parameters of Type “B”> runs in S5 as the following:

- “JU Logic Block” or as
- “A DB Data Block”.

In this case, automatic conversion is not possible because of missing type information in the formal parameter. Check your program for X instructions with parameters of type “B” and then convert these instructions manually.

Actual Parameters

For function blocks with parameters assigned, the S5/S7 Converter applies the actual parameters to block calls without changing them. If you have already defined addresses with an actual parameter, you will have to check this address definition and change it if necessary.

Examples:

- Defining a data word number:

  This must be converted into addressing done in bytes

- Defining an I/O address:

  The new module address must be used.

- Transferring a block:

  The block must include the new block number.

7.6 Standard Functions

S5 Standard Function Blocks

If your S5 program contains standard function blocks (SFBs), they are indicated as follows:

- Before conversion: by an asterisk in the “Std.” column of the dialog box “Converting S5 Files [<Name>ST.S5D]”, and
- After conversion: by displaying the message “Output not allowed (product no.)”.

The STEP 7 Standard software is supplied with S7 functions that have already been converted (former S5 standard function blocks) for floating-point math, signal functions, integrated functions, basic logic functions, and math functions with the names FC61 to FC125 (see Section 3.9).

Inserting FCs

To integrate the S7 functions into your S7 program, proceed as follows:

1. Open the project into which you want to insert the functions.
2. Open the standard library in the SIMATIC Manager with the converted S5 functions (StdLib30).
3. Copy the required S7 functions from the standard library into the S7 program.

Compiling the Program

8

Before you can run the converted and edited program, it must be compiled with the STL compiler. The procedure is exactly the same as for compiling a newly written text file.

Checking Data Consistency

Select the the menu command **File ▸ Consistency Check** to check the syntax and consistency of the source file at any time without causing blocks to be generated. Among other things, this function checks the following:

- The syntax,
- The symbols, and
- For the presence of called blocks in the program

Once the check is complete, a compiler report is generated which contains the name of the compiled file, the number of lines compiled, the number of errors present, as well as any warnings that occurred.

Compiling the Source File

Select the menu command **File ▸ Compile** to convert your source file into a block.

Once the compiling is complete, a compiler report is displayed containing any errors that occurred. This report is similar to the one displayed after a file has been checked for consistency. If a source file contains several blocks, then only the error-free ones are compiled and saved.

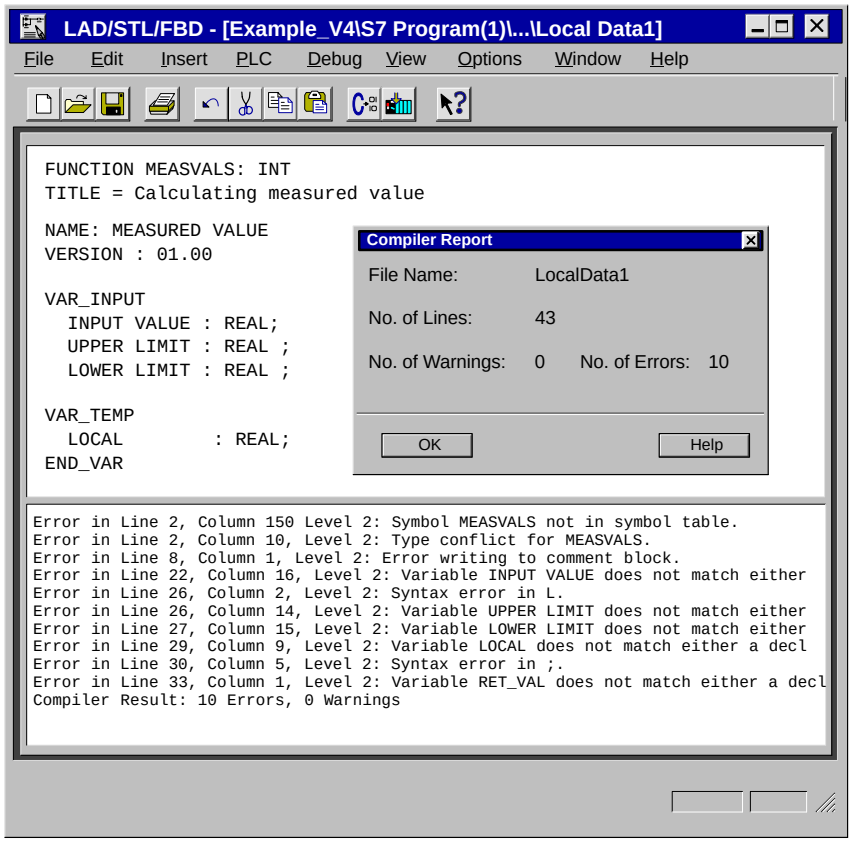
LAD/STL/FBD - [Example_V4\S7 Program(1)\...\Local Data1]
File Edit Insert PLC Debug View Options Window Help
FUNCTION MEASVALS: INT
TITLE = Calculating measured value
NAME: MEASURED VALUE
VERSION : 01.00
VAR_INPUT
INPUT VALUE : REAL;
UPPER LIMIT : REAL ;
LOWER LIMIT : REAL ;
VAR_TEMP
LOCAL : REAL;
END_VAR
Compiler Report
File Name: LocalData1
No. of Lines: 43
No. of Warnings: 0 No. of Errors: 10
OK
Help
Error in Line 2, Column 150 Level 2: Symbol MEASVALS not in symbol table.
Error in Line 2, Column 10, Level 2: Type conflict for MEASVALS.
Error in Line 8, Column 1, Level 2: Error writing to comment block.
Error in Line 22, Column 16, Level 2: Variable INPUT VALUE does not match either
Error in Line 26, Column 2, Level 2: Syntax error in L.
Error in Line 26, Column 14, Level 2: Variable UPPER LIMIT does not match either
Error in Line 27, Column 15, Level 2: Variable LOWER LIMIT does not match either
Error in Line 29, Column 9, Level 2: Variable LOCAL does not match either a decl
Error in Line 30, Column 5, Level 2: Syntax error in ;.
Error in Line 33, Column 1, Level 2: Variable RET_VAL does not match either a decl
Compiler Result: 10 Errors, 0 Warnings

Figure 8-1 Consistency Check and Compiling Source Files

Troubleshooting

If there are errors and/or warnings present in your converted program after it has been checked for consistency or compiled, they are listed under the source file in a second window section, along with their cause. If you then select an error message, the location of the corresponding error in the source file will be displayed. This coupling of error message with error location enables quick troubleshooting and error correction.

You can correct errors and make changes in overwrite mode. Press the INSERT key to toggle between the insert and overwrite modes.

Application Example

9

This chapter presents an application example illustrating four areas of operation that are either new in S7 or are now performed differently than in S5:

- Analog value processing
- Local data
- Evaluation of startup information in the organization blocks
- Block transfer

In this example, a motor operating to the right (clockwise) and left (counter-clockwise) is controlled by means of a digital I/O module. The speed is read by an analog input module and can be output by an analog output module. The digital and analog modules used in this example must be able to trigger a diagnostic interrupt.

Configuration

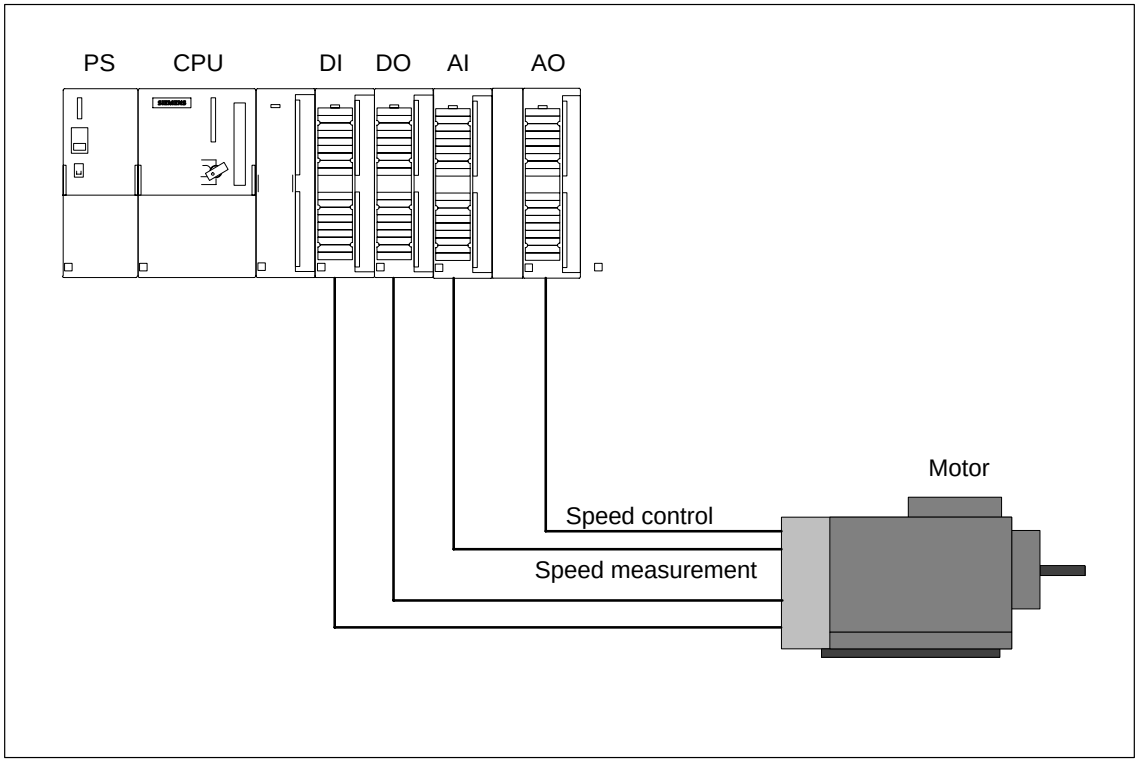

Figure 9-1 Configuration of the Application Example

9.1 Analog Value Processing

Conversion of Analog Values

The analog values are only processed in digital form by the CPU.

Analog input modules convert the analog processing signal into digital form.

Analog output modules convert the digital output value into an analog signal.

Analog Value Representation in S5

Table 9-1 Example of the Analog Input Module 6ES5 460-7LA13

Resolution	Analog Value															
Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Value	PS	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	A	E	O

The values for analog output modules are depicted as 12-bit twos complement.

Analog input modules can evaluate the value as a signed 12-bit number or as a 13-bit twos complement, as required.

The “O” bit indicates the amount of overflow.

The “E” bit is the error bit, which is set when an error occurs (for example, a wire break, if thius has been assigned parameters).

The “A” bit corresponds to the activity bit. If the bit is “0”, then the value displayed is valid.

Analog Value Representation in S7

For the same nominal range, the digitalized analog value is the same for input and output values.

Analog values are represented as a twos complement.

Table 9-2 Example of Analog Input Modules in S7

Resolution	Analog Value															
Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Value	S	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

The sign preceding (S = sign) the analog value is always in bit 15; here, a “0” stands for a positive and a “1” for a negative value.

In S7 there are no error bits.

If an error occurs, the value W#16#7FFF is output.

In the case of an error, blocks having diagnostic capability can trigger a diagnostic interrupt. The parameters for this interrupt are set in HWConfig.

If the resolution of an analog module is less than 15 bits, the analog value is left-aligned in the user data. Unoccupied low values have the signal state “0”.

Example

In this example, the speed of the motor is read by an analog input module having a resolution of 14 bits. This measured value has a bipolar range, such as +/–10V.

Upper and lower limits are transferred as parameters.

The analog value is checked for an upper and lower limit. If the value read lies outside of the permissible range, an error is reported using the binary result (BR = 0) and output as the value “0”. If the value is acceptable, it is output.

The analog value is output via the return value RET_VAL of the function. This RET_VAL is a functional value. In S7, this is a new functionality compared to S5.

```
FUNCTION FC1: REAL
TITLE = Analog Value Processing
NAME:         ANALOG
VERSION:      01.00
VAR_INPUT
       INPUT VALUE   : INT;        // Input value
       UPPER LIMIT   : REAL;       // Upper limit for the analog value
       LOWER LIMIT   : REAL;       // Lower limit for the analog value
END_VAR

BEGIN
NETWORK
TITLE = Checking Upper and Lower Limits
       O(;
       L      INPUT VALUE;  // Input value > Upper limit
       L      +27648;
       >I;
       );
       O(;                  // or
       L      INPUT VALUE;  // Input value < Lower limit
       L      -27648;
       <I;
       );
       NOT;
       L      0;
       JNB    END;          // If upper or lower limit exceeded, no further
                            // processing, return value = 0 and BR = “0”
                            // If no upper or lower limt exceeded => BR = “1”
NETWORK
TITLE = Converting Digital Value into Revolutions
       L      UPPER LIMIT;  // Formula for converting INPUT VALUE into
                            // revolutions:
       L      LOWER LIMIT;  // Analog value = (UPPER LIMIT - LOWER LIMIT)
                            //              * INPUT VALUE
       -R;                  //              / (55296 (number of units))
       L      INPUT VALUE;
       ITD;                 // Convert value into floating-point number
       DTR;
       *R;
       L      55296.0;
       /R;
END:   T      RET_VAL;
       BE;

END_FUNCTION
```

Figure 9-2 Analog Value Processing

9.2 Temporary Local Data

Temporary local data function as buffer storage and thus replace the scratchpad flags used in S5. Temporary local data can be used in all logic blocks. These data are lost after a logic block has been processed; they are located in the local data stack (L stack).

Example 1

This first example uses the temporary local data that are **symbolically** addressed as a buffer. A preset speed is converted into the digitalized measured value for the analog output module having a resolution of 14 bits. This measured value has a bipolar range, such as +/–10V.

Upper and lower limits are transferred as parameters.

The measured value is output via the return value (RET_VAL) for the function. Each function can optionally provide a return value. The data type of the return value is indicated in the description of the function. If no return value is provided, then the position for the data type has the entry VOID.

```
FUNCTION FC2: INT
TITLE = Calculating Measured Value
NAME:         MEASURED VALUE
VERSION:      01.00
VAR_INPUT
       INPUT VALUE   : REAL;       // Input value (current value)
       UPPER LIMIT   : REAL;       // Upper limit
       LOWER LIMIT   : REAL;       // Lower limit
END_VAR

VAR_TEMP
       LOCAL         : REAL;       // Local data as intermediate result
END_VAR
BEGIN
NETWORK
TITLE = Calculating Measured Value
       L      INPUT VALUE;         // Formula for calculating units:
       L      55296.0;             // Measured value = INPUT VALUE
       *R;                         //            * 55296 (number of units)
                                   //            / (UPPER LIMIT - LOWER LIMIT)
       T      LOCAL;               // Intermediate result in local data
       L      UPPER LIMIT;         // Buffer
       L      LOWER LIMIT;
       -R;
       L      LOCAL;
       TAK;
       /R;
       RND;                        // Convert floating-point number into integer
       T      RET_VAL;

END_FUNCTION
```

Figure 9-3 Calculating Measured Value

Example 2

The second example uses local data which are addressed **absolutely**, such as the S5 scratchpad flags, and shows how the clockwise and counter-clockwise operation of a motor is controlled. In this example, the input byte and the output byte are copied into the local data area. The user must reserve an area in the local stack for temporary local data use since the L stack is being used by the LAD/STL/FBD editor. The absolute addresses of the local data can be read in the block located in the declaration section. The local data bits are linked to each other by logic operations in the program. This produces the output signals which are written back at the end of the block to the output byte from the local data. The addresses for the input and output bytes can be assigned parameters.

Note

Inserting new variables in front of existing local data will cause the subsequent local data addresses to be shifted.

Table 9-3 Assignment of Inputs, Outputs, and Local Data

Address	Local Data	Name	Description
I n.0	L 0.0	ON	ON switch
I n.1	L 0.1	STOP	Stop motor
I n.2	L 0.2	EMERGENCY_STOP	Emergency stop button
I n.3	L 0.3	MOTOR_RIGHT	Motor: clockwise on
I n.4	L 0.4	MOTOR_LEFT	Motor: counter-clockwise on
I n.5	L 0.5	LIMIT SWITCH RIGHT	Limit switch, right
I n.6	L 0.6	LIMIT SWITCH LEFT	Limit switch, left
I n.7	L 0.7	-	Free
Q m.0	L 1.0	READY	Motor is ready
Q m.1	L 1.1	CLOCKWISE	Clockwise active
Q m.2	L 1.2	COUNTER-CLOCKWISE	Counter-clockwise active
Q m.3	L 1.3	POSITION REACHED	Position reached

Operation

The voltage is applied via the ON switch. The motor is now ready for use; this status is signaled by the output READY. The motor can be operated in a clockwise or counter-clockwise direction as required by using the buttons MOTOR_RIGHT and MOTOR_LEFT, respectively. The motor can only be operated in one direction at a time. Before changing the direction of motor rotation, the motor must be paused with the STOP switch. If a travel limit switch is reached, the motor is stopped. The EMERGENCY_STOP button also stops the motor; if this occurs then the motor can be restarted only after the EMERGENCY_STOP button has been reset.

```
FUNCTION FC3: VOID
TITLE = Motor Control
NAME:         MOTOR
VERSION:      01.00

VAR_INPUT
       INPUT BYTE    : BYTE;       // Input byte
END_VAR
VAR_IN_OUT
       OUTPUT BYTE   : BYTE;       // Output byte
END_VAR
VAR_TEMP
       IMAGE_INPUT BYTE     : BYTE;// Image of input byte
       IMAGE_OUTPUT BYTE    : BYTE;// Image of output byte
END_VAR

BEGIN
NETWORK
TITLE =  Motor Control
       L      INPUT BYTE;          // Copy input byte into local data area
       T      IMAGE_INPUT BYTE;
       L      OUTPUT BYTE;         // Copy output byte into local data area
       T      IMAGE_OUTPUT BYTE;
       ON     L0.0;                // Motor not switched on (no voltage)
       ON     L0.2;                // or EMERGENCY_STOP button pushed
       R      L1.0;                // => Motor is ready to reset
       R      L1.1;                // => Reset motor control
       R      L1.2;
       R      L1.3;                // => Reset position reached
       JC     END;                 // => No further signal evaluation
       A      L0.0;                // Motor switched on
       S      L1.0;                // => Set motor switched on
       A      L0.3;                // Operate motor clockwise
       AN     L0.4;                // Disable: no operation counter-clockwise
       AN     L1.2;                // and counter-clockwise not active
       FP     M0.0;                // Create positive edge
       S      L1.1;                // Then: switch on clockwise
       R      L1.3;                // Reset position reached
       A      L0.4;                // Operate motor counter-clockwise
       AN     L0.3;                // Disable: no operation clockwise
       AN     L1.1;                // and clockwise not active
       FP     M0.1;                // Create positive edge
       S      L1.2;                // Then: switch on counter-clockwise
       R      L1.3;                // Reset position reached
       O(;
       A      L0.5;                // Right limit switch reached and
       A      L1.1;                // clockwise active
       );
       O(;                         // or
       A      L0.6;                // Left limit switch reached and
       A      L1.2;                // counter-clockwise active
       );
       S      L1.3;                // => Position set reached
       O      L0.1;                // Stop motor switch pushed or
       O      L1.3;                // position reached
       R      L1.1;                // => Reset motor operation
       R      L1.2;
END:   L      IMAGE_OUTPUT BYTE;   // Copy local data to output byte
       T      OUTPUT BYTE;
END_FUNCTION
```

Figure 9-4 Motor Control Function

9.3 Evaluating the Startup Information from the Diagnostic Interrupt OB (OB82)

Startup Information

If the organization blocks are called by the operating system, the user is provided with system-wide startup information in the local data stack. This startup information is 20 bytes long and is available after OB processing has started.

Start Information for OB82

The startup information from the diagnostic interrupt OB contains the logical base address with four bytes of diagnostic information. The exact structure of this startup information is described in the *Reference Manual* **/235/**. Templates for the corresponding variable declaration table are located in the "StdLib30" standard library under the heading "StdOBs".

Based on diagnostic interrupt parameters previously configured in HWConfig, the digital modules make a request to the CPU for a diagnostic interrupt. This function applies to both incoming and outgoing events. After this request, the operating system calls the organization block OB82.

You can disable, delay, or re-enable the calling of the diagnostic interrupt OB with the help of the system functions (SFCs) 39 to 42. For further information, see the *Reference Manual* **/235/**.

Example

The following sample program shows how the external auxiliary voltage is evaluated. If the external auxiliary voltage is interrupted, the bit NO_EXT_VOLTAGE is set in DB82 "DB_DIAG". In addition, the module address and the time of the event are also saved. This information can be processed later in the program.

Before the STL source file is compiled, the symbol for the data block DB82 "DB_DIAG" must be entered in the symbol table.

```
DATA_BLOCK DB_DIAG
TITLE = Diagnostic Data
NAME:         DB_DIAG
VERSION:      01.00

STRUCT
       MDL_ADDR             : INT;        // Module address
       NO_EXT_VOLTAGE       : BOOL;       // No error bit for ext. aux. voltage
       DATE_TIME     : DATE_AND_TIME;     // Date and time at which the
                                          // diagnostic interrupt was triggered
       SFC_RET_VAL          : INT;        // Return code of SFC BLKMOV
END_STRUCT;

BEGIN
END_DATA_BLOCK

ORGANIZATION_BLOCK OB82
TITLE = Diagnostic Interrupt
NAME:         Diagnostic
VERSION:      01.00
VAR_TEMP
       OB82_EV_CLASS        : BYTE;// Event class and IDs:
                                   // B#16#38: outgoing event
                                   // B#16#39: incoming event
       OB82_FLT_ID          : BYTE;// Error code (B#16#42)
       OB82_PRIORITY        : BYTE;// Priority class 26 or 28
       OB82_OB_NUMBR        : BYTE;// OB number
       OB82_RESERVED_1      : BYTE;// Reserved
       OB82_IO_FLAG         : BYTE;// Input module: B#16#54
                                   // Output module: B#16#55
       OB82_MDL_ADDR        : INT; // Logical base address of module
                                   // where the fault occurred
       OB82_MDL_DEFECT      : BOOL;// Module is defective
       OB82_INT_FAULT       : BOOL;// Internal fault
       OB82_EXT_FAULT       : BOOL;// External fault
       OB82_PNT_INFO        : BOOL;// Channel fault
       OB82_EXT_VOLTAGE     : BOOL;// External voltage failed
       OB82_FLD_CONNCTR     : BOOL;// Front panel connector not plugged
       OB82_NO_CONFIG       : BOOL;// Module is not configured
       OB82_CONFIG_ERR      : BOOL;// Incorrect parameters on module
       OB82_MDL_TYPE        : BYTE;// Bit 0 to 3: Module class
                                   // Bit 4: Channel information exists
                                   // Bit 5: User information exists
                                   // Bit 6: Diag. interrupt from substitute
                                   // Bit 7: Reserve
       OB82_SUB_MDL_ERR     : BOOL;// Submodule is missing or has an error
       OB82_COMM_FAULT      : BOOL;// Communication problem
       OB82_MDL_STOP : BOOL;       // Operating mode (0: RUN, 1: STOP)
       OB82_WTCH_DOG_FLT    : BOOL;// Watchdog timer responded
       OB82_INT_PS_FLT      : BOOL;// Internal power supply failed
       OB82_PRIM_BATT_FLT   : BOOL;// Battery dead
       OB82_BCKUP_BATT_FLT  : BOOL;// Entire backup failed
       OB82_RESERVED_2      : BOOL;// Reserved
       OB82_RACK_FLT        : BOOL;// Rack failure
       OB82_PROC_FLT        : BOOL;// Processor failure
       OB82_EPROM_FLT       : BOOL;// EPROM fault
       OB82_RAM_FLT         : BOOL;// RAM fault
```

continued

Figure 9-5 Diagnostic Data Evaluation

```
       OB82_ADC_FLT         : BOOL;       // ADC/DAC error
       OB82_FUSE_FLT        : BOOL;       // Fuse blown
       OB82_HW_INTR_FLT     : BOOL;       // Hardware interrupt lost
       OB82_RESERVED_3      : BOOL;       // Reserved
       OB82_DATE_TIME: DATE_AND_TIME;     // Date and time when OB was called
END_VAR

BEGIN
NETWORK
TITLE = Diagnostic Interrupt
       L      OB82_MDL_ADDR;              // Save module address
       T      DB_DIAG.MDL_ADDR;
       L      OB82_EV_CLASS;              // Event class = B#16#38:
       L      B#16#38;                    // Outgoing event
       ==I;
       JC     GO;
                                          // Incoming event:
       A      OB82_EXT_VOLTAGE;           // Check if no ext. auxiliary voltage
       S      DB_DIAG.NO_EXT_VOLTAGE;     // Set bit
       JU     TIME;
                                          // Outgoing event:
GO:    A      OB82_EXT_VOLTAGE;           // Ext. auxiliary voltage present
again
       R      DB_DIAG.NO_EXT_VOLTAGE;     // Reset bit

NETWORK
TITLE = Save Time
TIME:  CALL   SFC20(                      // SFC BLKMOV
       SRCBLK :=OB82_DATE_TIME,           // Save date and time at which
       RET_VAL:=DB_DIAG.SFC_RET_VAL,      // diagnostic interrupt
       DSTBLK :=DB_DIAG.DATE_TIME);       // was requested
END_ORGANIZATION_BLOCK
```

Figure 9-6 Diagnostic Data Evaluation, continued

9.4 Block Transfer

You can use the system function SFC20 “BLKMOV” (block move) to copy the contents of one memory area, the “source field”, into another memory area, the “target field”.

You can use SFC20 “BLKMOV” to copy all inputs, outputs, bit memory, and data.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
SRCBLK	INPUT	ANY	I, Q, M, D, L	Indicates the memory area to be copied (source field).
RET_VAL	OUTPUT	INT	I, Q, M, D, L	If an error occurs during processing of the function, the return value will contain an error code.
DSTBLK	OUTPUT	ANY	I, Q, M, D, L	Indicates the memory area into which the data is to be copied (target field)

Note

The source and target fields must not overlap. If the target field specified is larger than the source field, than only the amount of data contained in the source field is copied into the target field.

If the target field specified is smaller that the source field, then only the amount of data that the target field can accept is copied.

If you want to have the parameters for the source and target areas of SFC20 “BLKMOV” filled with variable values instead of constant pointers, you can do this by using temporary variables of the data type ANY.

Structure of the ANY Pointer for Data Types

The following tables show the structure of the ANY pointer.

Table 9-4 ANY Pointer

Byte n	Byte n+1	Byte n+2	Byte n+3	Byte n+4	Byte n+5	Byte n+6	Byte n+7	Byte n+8	Byte n+9
B#16#10	Type *(see Table 9-5)*	Length		Data block no. for data blocks		Area pointer *(see Figure 9-7)*			

Table 9-5 Type (Byte n+1)

Value:	01	02	03	04	05	06	07
Type:	BOOL	BYTE	CHAR	WORD	INT	DWORD	DINT
Value:	**08**	**09**	**0A**	**0B**	**0C**	**0E**	**13**
Type:	REAL	DATE	TOD	TIME	S5TIME	DT	String

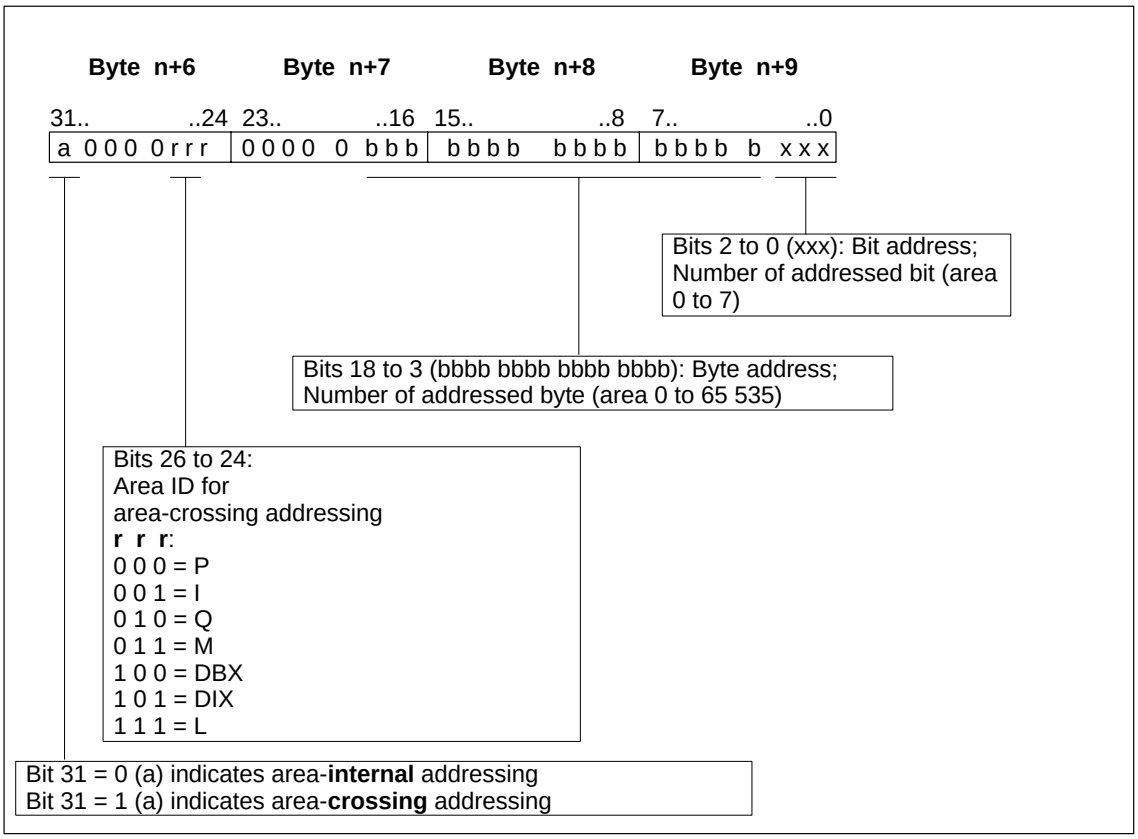
Byte n+6
Byte n+7
Byte n+8
Byte n+9
31.. ..24 23.. ..16 15.. ..8 7.. ..0
a 0 0 0 0 r r r
0 0 0 0 0 b b b
b b b b b b b b
b b b b b x x x
Bits 2 to 0 (xxx): Bit address; Number of addressed bit (area 0 to 7)
Bits 18 to 3 (bbbb bbbb bbbb bbbb): Byte address; Number of addressed byte (area 0 to 65 535)
Bits 26 to 24:
Area ID for
area-crossing addressing
r r r:
0 0 0 = P
0 0 1 = I
0 1 0 = Q
0 1 1 = M
1 0 0 = DBX
1 0 1 = DIX
1 1 1 = L
Bit 31 = 0 (a) indicates area-internal addressing
Bit 31 = 1 (a) indicates area-crossing addressing

Figure 9-7 Area Pointer (Byte n+6 to Byte n+9)

Example

The example shows a function which uses the the system function SFC20 “BLKMOV” to copy data area (in data blocks). Variable source and target areas can be entered as parameters.

Principle

The function contains two ANY pointers in the local data area and one ANY pointer for the target area. As a rule, the ANY data type can only be used for variables in the local data area.

In the function, the ANY pointer is assigned a value as indicated in the structure previously described. This value is indicated in the parameters when the SFC20 “BLKMOV” is called.

```
FUNCTION FC4: INT
TITLE = Copying Data Areas
NAME:         COPY
VERSION:      01.00

VAR_INPUT
       SOURCE_DBNO   : INT;        // DB no. of source area
       SOURCE_BEGIN  : INT;        // Data word no. of beginning of source area
       SOURCE_LENGTH : INT;        // Length of source area in bytes
       DEST_DBNO     : INT;        // DB no. of destination area
       DEST_BEGIN    : INT;        // Data word no. of beginning of dest. area
       DEST_LENGTH   : INT;        // Length of destination area in bytes
END_VAR

VAR_TEMP
       POINTER_SOURCE: ANY;        // ANY pointer for the source area
       POINTER_DEST  : ANY;        // ANY pointer for the destination area
END_VAR

BEGIN
NETWORK
TITLE = Preparing Source Pointer
       L      P##POINTER_SOURCE;   // Load address of pointer in source area
       LAR1;                       // into address register 1
       L      W#16#1002;           // Write area ID for data area in
       T      LW[AR1, P#0.0];      // ANY pointer for source
       L      SOURCE_DBNO;         // Write DB no. in ANY pointer for source
       T      LW[AR1, P#4.0];
       L      SOURCE_BEGIN;        // Convert beginning of data area
       SLD    3;                   // into pointer format,
       OD     DW#16#84000000;      // Link area ID
       T      LD[AR1, P#6.0];      // and write in ANY pointer for source
       L      SOURCE_LENGTH;       // Write length of data area in ANY pointer
       T      LW[AR1, P#2.0];      // for source
```

continued

Figure 9-8 Copying Data Areas

```
NETWORK
TITLE = Preparing Destination Pointer
       L      P##POINTER_DEST;     // Load address of pointer to dest. area
       LAR1;                       // in address register 1
       L      W#16#1002;           // Write area ID for data area in
       T      LW[AR1, P#0.0];      // ANY pointer for destination
       L      DEST_DBNO;           // DB no. in ANY pointer for destination
       T      LW[AR1, P#4.0];
       L      DEST_BEGIN;          // Convert beginning of data area
       SLD    3;                   // into pointer format
       OD     DW#16#84000000;      // Link area ID
       T      LD[AR1, P#6.0];      // and write in ANY pointer for destination
       L      DEST_LENGTH;         // Write length of data area to ANY pointer
       T      LW[AR1, P#2.0];      // for destination
NETWORK
TITLE = Copying Data
       CALL   SFC 20(              // Copy data with SFC BLKMOV (block transfer)
       SRCBLK := POINTER_SOURCE,   // Pointer to source area
       RET_VAL:= RET_VAL,          // Return code of SFC BLKMOV
       DSTBLK := POINTER_DEST);    // Pointer to destination area
END_FUNCTION
```

Figure 9-9 Copying Data Areas, continued

9.5 Calling the Examples

This section contains the symbol table, the data blocks required for assigning values to the block parameters, and the organization block OB1 with the calls for the functions previously described.

Table 9-6 Symbol Table

Symbol	Address	Data Type	Comments
DB_DIAG	DB82	DB82	Diagnostic data block
DB_MEASVALS	DB100	DB100	Data block for measured values
DB_MOTOR_1	DB110	DB110	Data block for motor 1
ERROR	MW 100	WORD	Return value of the function FC4 for block transfer

```
DATA_BLOCK DB_MEASVALS
TITLE = Measured Values
NAME:         DB_MEASVALS
VERSION:      01.00
STRUCT
       ANALOGVAL_1   : REAL;       // Analog value 1 from FC1
       ANALOGVAL_2   : REAL;       // Analog value 2 from FC2
       DIGITALVAL_2  : INT;        // Digitalized measured value from FC2
END_STRUCT;
BEGIN
END_DATA_BLOCK

DATA_BLOCK DB_MOTOR_1
TITLE = Motor Data
NAME:         DB_MOTOR_1
VERSION:      01.00
STRUCT
       CONTROL WORD  : WORD;       // Control of motor 1
       SPEED         : REAL;       // Speed of motor 1
       TEMPERATURE   : REAL;       // Temperature of motor 1
       CURRENT       : REAL;       // Current consumption of motor 1
END_STRUCT;
BEGIN
END_DATA_BLOCK

ORGANIZATION_BLOCK OB1
TITLE = Call in Cycle
NAME:         CYCLE
VERSION:      01.00
VAR_TEMP
  STARTINFO: ARRAY [1..20] of BYTE;
END_VAR
BEGIN
NETWORK
TITLE = Call of Functions
CALL   FC1(                        // Call function for
       INPUT VALUE   := IW 0,      // analog value processing
       UPPER LIMIT   := +10.0,     // Measured range: +/–10V
       LOWER LIMIT   := -10.0,
       RET_VAL       := DB_MEASVALS.ANALOGVAL_1);
                                   // RET_VAL = Analog value
                                   // Call function for calculating
CALL   FC2(                        // digitalized measured value
       INPUT VALUE   := DB_MEASVALS.ANALOGVAL_2,//
       UPPER LIMIT   := +10.0,     // Measured range: +/-10V
       LOWER LIMIT   := -10.0,
       RET_VAL       := DB_MEASVALS.DIGITALVAL_2);
                                   // RET_VAL = digitalized meas. value
CALL   FC3(                        // Call function for motor control
       INPUT BYTE    := IB 4,
       OUTPUT BYTE   := QB 8);
CALL   FC4(                        // Call function for block transfer
       SOURCE_DBNO   := 100,       // Source: DB100
       SOURCE_BEGIN  := 0,         // From data byte DBB 0
       SOURCE_LENGTH := 8,         // Length: 4 Byte
       DEST_DBNO     := 110,       // Destination: DB110
       DEST_BEGIN    := 2,         // From data byte DBB 6
       DEST_LENGTH   := 8,         // Length: 4 bytes
       RET_VAL       := ERROR);    // RET_VAL = Error code for SFC20 BLKMOV
END_ORGANIZATION_BLOCK
```

Figure 9-10 OB1

Appendix

Address and Instruction Lists A

Literature List B

Glossary, Index

Address and Instruction Lists

A

A.1 Addresses

Convertible Addresses

The following addresses are converted:

Table A-1 Convertible Addresses

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”A”	”Q”	”A”	”Q”
”AB”	”QB”	”AB”	”QB”
”AD”	”QD”	”AD”	”QD”
”AW”	”QW”	”AW”	”QW”
”BF”	”BN”	””	””
”D”	”D”	”DBX”	”DBX”
”DW”	”DW”	”DBW”	”DBW”
”DD”	”DD”	”DBD”	”DBD”
”DR”	”DR”	”DBB”	”DBB”
”DL”	”DL”	”DBB”	”DBB”
”E”	”I”	”E”	”I”
”EB	”IB”	”EB”	”IB”
”ED”	”ID”	”ED”	”ID”
”EW”	”IW”	”EW”	”IW”
”M”	”F”	”M”	”M”
”MB”	”FY”	”MB”	”MB”
”MD”	”FD”	”MD”	”MD”
”MW”	”FW”	”MW”	”MW”
”PW”	”PW”	”PEW/PAW”	”PIW/PQW”
”PY”	”PY”	”PEB/PAB”	”PIB/PQB”
”QB”	”OY”	”PEB/PAB”	”PIB/PQB”
”QW”	”OW”	”PEW/PAW”	”PIW/PQW”
”S”	”S”	”M”	”M”
”SD”	”SD”	”MD”	”MD”
”SW”	”SW”	”MW”	”MW”

Table A-1 Convertible Addresses

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”SY”	”SY”	”MB”	”MB”
”T”	”T”	”T”	”T”
”Z”	”C”	”Z”	”C”
”= <Formal parameter>”	”= <Formal parameter>”	”# <Formal parameter>”	”# <Formal parameter>”

Non-Convertible Addresses

Table A-2 shows the addresses that **cannot be converted**.

Table A-2 Non-Convertible Addresses

S5 STL (German)	S5 STL (International)
”A1”	”A1”
”A2”	”A2”
”BA”	”RI”
”BB”	”RJ”
”BR”	”BR”
”BS”	”RS”
”BT”	”RT”
”CB”	”CY”
”CD”	”CD”
”CW”	”CW”
”GB”	”GY”
”GD”	”GD”
”GW”	”GW”
”SA”	”SA”

A.2 Instructions

Conversion Instructions without Addresses

Table A-3 shows all the S5 instructions (without addresses) in STL that can be converted automatically into S7 STL:

Table A-3 Convertible Instructions (without Addresses)

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”AF”	”RA”	”CALL SFC42”	”CALL SFC42”
”AS”	”IA”	”CALL SFC41”	”CALL SFC41”
”BEA”	”BEU”	”BEA”	”BEU”
”BEB”	”BEC”	”BEB”	”BEC”
”+D”	”+D”	”+D”	”+D”
”–D”	”–D”	”–D”	”–D”
”!=D”	”!=D”	”==D”	”==D”
”><D”	”><D”	”<>D”	”<>D”
”>D”	”>D”	”>D”	”>D”
”>=D”	”>=D”	”>=D”	”>=D”
”<D”	”<D”	”<D”	”<D”
”<=D”	”<=D”	”<=D”	”<=D”
”DED”	”DED”	”BTD”	”BTD”
”DEF”	”DEF”	”BTI”	”BTI”
”DUD”	”DUD”	”DTB”	”DTB”
”DUF”	”DUF”	”ITB”	”ITB”
”ENT”	”ENT”	”ENT”	”ENT”
”+F”	”+F”	”+I”	”+I”
”–F”	”–F”	”–I”	”–I”
”:F”	”:F”	”/I”	”/I”
”xF”	”xF”	”*I”	”*I”
”!=F”	”!=F”	”==I”	”==I”
”><F”	”><F”	”<>I”	”<>I”
”>F”	”>F”	”>I”	”>I”
”>=F”	”>=F”	”>=I”	”>=I”
”<F”	”<F”	”<I”	”<I”
”<=F”	”<=F”	”<=I”	”<=I”
”FDG”	”FDG”	”DTR”	”DTR”
”+G”	”+G”	”+R”	”+R”
”–G”	”–G”	”–R”	”–R”
”:G”	”:G”	”/R”	”/R”
”xG”	”xG”	”*R”	”*R”

Table A-3 Convertible Instructions (without Addresses), continued

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”!=G”	”!=G”	”==R”	”==R”
”><G”	”><G”	”<>R”	”<>R”
”>G”	”>G”	”>R”	”>R”
”>=G”	”>=G”	”>=R”	”>=R”
”<G”	”<G”	”<R”	”<R”
”<=G”	”<=G”	”<=R”	”<=R”
”GFD”	”GFD”	”RND”	”RND”
”KEW”	”CFW”	”INVI”	”INVI”
”KZD”	”CSD”	”NEGD”	”NEGD”
”KZW”	”CSW”	”NEGI”	”NEGI”
”O”	”O”	”O”	”O”
”O(”	”O(”	”O(”	”O(”
”OW”	”OW”	”OW”	”OW”
”STP”	”STP”	”CALL SFC 46”	”CALL SFC 46”
”STS”	”STS”	”CALL SFC 46”	”CALL SFC 46”
”STW”	”STW”	”CALL SFC 46”	”CALL SFC 46”
”TAK”	”TAK”	”TAK”	”TAK”
”U(”	”A(”	”U(”	”A(”
”UW”	”AW”	”UW”	”AW”
”XOW”	”XOW”	”XOW”	”XOW”
”)”	”)”	”)”	”)”
”***”	”***”	”NETWORK”	”NETWORK”

Conversion Instructions with Addresses

Table A-4 shows all the S5 instructions (with addresses) in STL that can be converted automatically into S7 STL:

Table A-4 Convertible Instructions (with Addresses)

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”A”	”C”	”AUF”	”OPN”
”ADD BF” ”ADD DH” ”ADD KF”	”ADD BF” ”ADD DH” ”ADD KF”	”+” ”+” ”+”	”+” ”+” ”+”
”AX”	”CX”	”AUF”	”OPN”
”B”	”DO”	”Instruction sequence for indirect addressing”	”Instruction sequence for indirect addressing”

Table A-4 Convertible Instructions (with Addresses), continued

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”BA”	”BA”	””	””
”BAB”	”DOC”	”SPB”	”JC”
”D”	”D”	”DEC”	”DEC”
”E”	”G”	”CALL SFC22”	”CALL SFC22”
”EX”	”GX”	”CALL SFC22”	”CALL SFC22”
”FR”	”FR”	”FR”	”FR”
”I”	”I”	”INC”	”INC”
”L”	”L”	”L”	”L”
”LC”	”LD”	”LC”	”LC”
”NOP”	”NOP”	”NOP”	”NOP”
”O”	”O”	”O”	”O”
”ON”	”ON”	”ON”	”ON”
”P”	”TB”	”SET; U”	”SET; A”
”PN”	”TBN”	”SET; UN”	”SET; AN”
”R”	”R”	”R”	”R”
”RB”	”RB”	”R”	”R”
”RD”	”RD”	”R”	”R”
”RLD”	”RLD”	”RLD”	”RLD”
”RLW”	”RLW”	”RLW”	”RLW”
”RRD”	”RRD”	”RRD”	”RRD”
”RRW”	”RRW”	”RRW”	”RRW”
”RU”	”RU”	”SET; R”	”SET; R”
”S”	”S”	”S”	”S”
”SA”	”SF”	”SA”	”SF”
”SAR”	”SFD”	”SA” Timer ”ZR” Zähler	”SF” Timer ”CD” Counter
”SE”	”SD”	”SE”	”SD”
”SI”	”SP”	”SI”	”SP”
”SLD”	”SLD”	”SLD”	”SLD”
”SLW”	”SLW”	”SLW”	”SLW”
”SPA”	”JU”	”SPA”	”JU”
”SPB”	”JC	”SPB”	”JC”
”SPM”	”JM”	”SPM”	”JM”
”SPN”	”JN”	”SPN”	”JCN”
”SPO”	”JO”	”SPO”	”JO”
”SPP”	”JP”	”SPP”	”JP”

Table A-4 Convertible Instructions (with Addresses), continued

S5 STL (German)	S5 STL (International)	S7 STL (German)	S7 STL (International)
”SPR”	”JUR”	”SPA”	”JU”
”SPS”	”JOS”	”SPS”	”JOS”
”SPZ”	”JZ”	”SPZ”	”JZ”
”SRD”	”SRD”	”SRD”	”SRD”
”SRW”	”SRW”	”SRW”	”SRW”
”SS”	”SS”	”SS”	”SS”
”SSV”	”SSU”	”SS” Timer ”ZV” Zähler	”SS” Timer ”CU” Counter
”SU”	”SU”	”SET; S”	”SET; S”
”SV”	”SE”	”SV”	”SE”
”SVD”	”SSD”	”SSD”	”SSD”
”SVW”	”SSW”	”SSI”	”SSI”
”SVZ”	”SEC”	”SV” Timer ”S” Zähler	”SE” Timer ”S” Counter
”T”	”T”	”T”	”T”
”TNB”	”TNB”	”CALL SFC20”	”CALL SFC20”
”TNW”	”TNW”	”CALL SFC20”	”CALL SFC20”
”U”	”A”	”U”	”A”
”UN”	”AN”	”UN”	”AN”
”ZR”	”CD”	”ZR”	”CD”
”ZV”	”CU”	”ZV”	”CU”
”=”	”=”	”=”	”=”

Non-Convertible Instructions

The following table shows the S5 STL instructions that cannot be converted automatically.

Table A-5 Non-Convertible Instructions

S5 STL (German)	S5 STL (International)
”AAS”	”IAI”
”AAF”	”RAI”
”ABR”	”ABR”
”ACR”	”ACR”
”AFF”	”RAE”
”AFS”	”IAE”
”ASM”	”ASM”
”BAF”	”BAF”

Table A-5 Non-Convertible Instructions

S5 STL (German)	S5 STL (International)
”BAS”	”BAS”
”BI” (can only be converted for parameter type D/constant)	”DI” (can only be converted for parameter type D/constant)
”BLD”	”BLD”
”LB”	”LB”
”LD”	”LD”
”LD=<Formal parameter>” (can only be converted for parameter type D/constant)	”LD=<Formal parameter>” (can only be converted for parameter type D/constant)
”LDI”	”LDI”
”LIM”	”LIM”
”LIR”	”LIR”
”LRB”	”LRB”
”LRD”	”LRD”
”LRW”	”LRW”
”LW”	”LW”
”LW=<Formal parameter>” (can only be converted for parameter type D/constant)	”LW=<Formal parameter>” (can only be converted for parameter type D/constant)
”MA1”	”MA1”
”MAB”	”MAB”
”MAS”	”MAS”
”MBA”	”MBA”
”MBR”	”MBR”
”MBS”	”MBS”
”MSA”	”MSA”
”MSB”	”MSB”
”SEF”	”SEE”
”SES”	”SED”
”SIM”	”SIM”
”TB”	”TB”
”TDI”	”TDI”
”TIR”	”TIR”
”TSC”	”TSC”
”TSG”	”TSG”
”TRB”	”TRB”
”TRD”	”TRD”
”TRW”	”TRW”
”TW”	”TW”
”TXB”	”TXB”

Table A-5 Non-Convertible Instructions

S5 STL (German)	S5 STL (International)
”TXW”	”TXW”
”UBE”	”UBE”

Literature List

B

/21/ Technical Overview: *S7/M7 Programmable Controllers,*
Distributed I/O with PROFIBUS-DP and AS-i

/30/ Primer: *S7-300 Programmable Controller,*
Quick Start

/70/ Manual: *S7-300 Programmable Controller,*
Hardware and Installation

/71/ Reference Manual: *S7-300 and M7-300 Programmable Controllers,*
Module Specifications

/72/ Instruction List: *S7-300 Programmable Controller,*
CPU 312 IFM, 314 IFM, 313, 314, 315-2DP

/100/ Manual: *S7-400, M7-400 Programmable Controllers,*
Hardware and Installation

/101/ Reference Manual: *S7-400, M7-400 Programmable Controllers,*
Module Specifications

/102/ Reference Guide: *S7-400 Instruction List,*
CPU 412, 413, 414, 416

/231/ User Manual: *Standard Software for S7 and M7,*
STEP 7

/232/ Manual: *Statement List (STL) for S7-300 and S7-400,*
Programming

/233/ Manual: *Ladder Logic (LAD) for S7-300 and S7-400,*
Programming

/234/ Programming Manual: *System Software for S7-300 and S7-400,*
Program Design

/235/ Reference Manual: *System Software for S7-300 and S7-400,*
System and Standard Functions

/236/ Manual: *Function Block Diagram (FBD) for S7-300 and S7-400,*
Programming

/249/ Manual: *Continuous Function Chart (CFC),*
Volume 2: *S7/M7*

/250/ Manual: *Structured Control Language (SCL) for S7-300 and S7-400,*
Programming

/251/ Manual: *GRAPH for S7-300 and S7-400,*
Programming Sequential Control Systems

/252/ Manual: *HiGraph for S7-300 and S7-400,*
Programming State Graphs

/253/ Manual: *C Programming for S7-300 and S7-400,*
Writing C Programs

/254/ Manual: *Continuous Function Chart* (CFC),
Volume1

/270/ Manual: *S7-PDIAG for S7-300 and S7-400,*
Configuring Process Diagnostics for LAD, STL, and FBD

/271/ Manual: *NETPRO,*
Configuring Networks

/280/ Programming Manual: *System Software for M7-300 and M7-400,*
Program Design

/281/ Reference Manual: *System Software for M7-300 and M7-400,*
System and Standard Functions

/282/ User Manual: *System Software for M7-300 and M7-400,*
Installation and Operation

/290/ User Manual: *ProC/C++ for M7-300 and M7-400,*
Writing C Programs

/291/ User Manual: *ProC/C++ for M7-300 and M7-400,*
Debugging C Programs

/500/ Manual: *SIMATIC NET,*
NCM S7 for Industrial Ethernet

/501/ Manual: *SIMATIC NET,*
NCM S7 for PROFIBUS

/800/ *DOCPRO*
Creating Documentation (CD only)

/801/ *TeleService for S7, C7, and M7*
Remote Maintenance for Automation Systems (CD only)

/802/ *PLC Simulation for S7-300 and S7-400* (CD only)

/803/ Reference Manual: *Standard Software for S7-300 and S7-400,*
STEP 7 Standard Functions, Part 2 (CD only)

Glossary

A

Actual Parameter

Actual parameters replace formal parameters when a function block (FB) or function (FC) is called, for example, the formal parameter “START” is replaced by the actual parameter “I3.6”.

Address

An address includes the address identifier and the physical memory location where the address is stored. Examples: Input I12.1; Memory Word MW25; Data Block DB3.

An address is part of a STEP 7 statement and specifies what the processor should execute the instruction on. Addresses can be absolute or symbolic.

Assigning Parameters

Assigning parameters means setting the behavior of a module.

B

Block

Blocks are part of the user program and can be distinguished by their function, their structure, or their purpose. STEP 7 provides the following types of blocks:

- Logic blocks (FB, FC, OB, SFB, SFC)
- Data blocks (DB, SDB)
- User-defined data types (UDT)

Block Call

A block call is the branch into the called block taken during program processing.

Block Parameter

Block parameters are token values within multipurpose blocks which are supplied with current values when the corresponding block is called.

C

Compiler

The compiler program for compiling a program written in a higher programming language to the machine code the CPU uses is known as a compiler.

Configuring

Configuring is the selection and putting together of the individual components of a programmable logic controller (PLC), and the installation of the required software and adapting it to the specific task (such as assigning parameters to the modules.)

D

Data Block (DB)

Data blocks are areas in the user program which contain user data. There are shared data blocks which can be accessed by all logic blocks, and there are instance data blocks which are associated with a particular function block (FB) call. Data blocks contain no logic instructions, in contrast to all other types of block.

Data, Static

Static data are local data in a function block which are stored in the instance data block and thus remain stored until the next function block call.

Data, Temporary

Temporary data are local data in a block which are kept in the L stack while the block is in use and are no longer available once the block is closed.

Data Type

With the help of data types you can specify how the value of a variable or a constant is to be used in the user program. There are two data types according to IEC 1131-3 available to users of SIMATIC S7: elementary and complex data types.

Data Type, Complex

Complex data types are defined by the user with the data type declaration. They do not have their own name and cannot be used more than once. A distinction is made between arrays and structures. The data types *String* and *Date and Time* also belong to this category.

Data Type, Elementary

Elementary data types are predefined data types according to IEC 1131-3, for example, the data type BOOL defines a binary variable (“bit”); the data type INT defines a 16-bit fixed-point variable (integer).

Declaration Section

The local data of a logic block are declared in the declaration section if the program is generated using a text editor.

F

Formal Parameter

A formal parameter is a token value for the “actual parameter” of logic blocks which can be assigned parameters. The formal parameters are declared by the user in the case of function blocks and functions, but are already present in the case of system function blocks and system functions.

When calling the block, an actual parameter is assigned to the formal parameter so that the called block works with its current value. The formal parameters are included amongst the local data of the block and are divided into input, output, and I/O parameters.

Function (FC)

According to the International Electrotechnical Commission’s IEC 1131–3 standard, functions are logic blocks that do not reference an instance data block, meaning they do not have a ’memory’. A function allows you to pass parameters in the user program, which means they are suitable for programming complex functions that are required frequently, for example, calculations. As there is no memory available, the calculated values must be processed immediately following the FC call.

Function Block (FB)

According to the International Electrotechnical Commission’s IEC 1131–3 standard, function blocks are logic blocks that reference an instance data block, meaning they have static data. A function block allows you to pass parameters in the user program, which means they are suitable for programming complex functions that are required frequently, for example, control systems, operating mode selection. As function blocks have a ’memory’ in the form of the associated instance data block, its parameters (outputs, for example) can be accessed at any time and any point in the user program.

I

I/O, Distributed (DP)

The distributed I/O consists of analog and digital modules which are located at a physical distance from the central rack. Characteristic of the distributed I/O is the modular rack system whose aim it is to save connecting wires, thereby saving costs by placing the I/O modules close to the process.

Instance

An “instance” is the call of a function block; an instance data block is associated with this call.

Instance Data Block

An instance data block stores the formal parameters and static data for function blocks. An instance data block can be associated with a function block call or a call hierarchy of function blocks.

Instruction

An instruction is part of a STEP 7 statement and specifies what the processor should do.

L

Local Data

Local data are data assigned to a logic block which are declared in its declaration section or its variable declaration. They cover (depending on the block): formal parameters, static data, temporary data.

Logic Block

In SIMATIC S7, a logic block is a block that contains part of the STEP 7 user program. The other type of block is a data block which contains only data. The following list shows the types of logic blocks:

- Organization block (OB)
- Function block (FB)
- Function (FC)
- System function block (SFB)
- System function (SFC)

M

Macro

A macro is a sequence of instructions which are combined into a mnemonic call optimized for execution.

O

Online Help

STEP 7 enables you to display context-sensitive help on the screen while you are working with the programming software.

Organization Block (OB)

Organization blocks form the interface between the CPU operating system and the user program. The sequence in which the user program should be processed is laid down in the organization blocks.

P

Pointer

A pointer is a variable which does not possess a particular value but the address of another variable. With pointer instructions, the type on the right side of the operator must correspond to the type on the left side.

Programming Language

A programming language is used to create user programs and provides a specific ’vocabulary’ for this purpose in the form of text instructions or graphic elements. These instructions are entered by the user using an editor and compiled into an executable user program.

Project

A project is a container for all objects in an automation task, independent of the number of stations, modules, and how they are connected in a network.

R

Retentive

Data are called retentive if they have the same value after a power supply failure as before the power supply failed. The data are backed up in two ways:

- Voltage backup
- Backup memory

S

S7 Program

An S7 program is a container for blocks, source files, and charts for S7 programmable modules which also contains the symbol table.

Shared Data

Shared data are data which can be accessed from any logic block (function (FC), function block (FB), organization block (OB)). These are bit memory (M), inputs (I), outputs (Q), timers (T), counters (C), and elements of data blocks (DB). You can access shared data either absolutely or symbolically.

Statement

A statement is the smallest independent part of a user program created in a textual language. It represents a command for the processor.

Statement List (STL)

Statement List is a textual representation of the STEP 7 programming language, similar to machine code.

Symbol

A symbol is a name defined by the user, taking syntax rules into consideration. This name can be used in programming and in operating and monitoring once you have defined it (for example, as a variable, a data type, a jump label, or a block).

Example: Address: I5.0, Data Type: BOOL, Symbol: Emer_Off_Switch

A distinction is made between shared symbols and block-specific symbols. Shared symbols are available to all parts of the program, therefore the symbol you assign must be unique for the whole user program. Block-specific symbols are only recognized within the block for which they were assigned.

Symbol Table

A table used to assign symbols (or symbolic names) to addresses for shared data and blocks.
Examples: Emer_Off (Symbol), I1.7 (Address)
Controller (Symbol), SFB24 (Block)

V

Variable

A variable defines an item of data with variable content which can be used in the STEP 7 user program. A variable consists of an address and a data type, and can be identified by means of a symbolic name.

Index

A

Absolute address, 4-3
Accumulator instructions, 3-35
Actuator-sensor interface, 2-10
Adapter casing, 2-13
Address
  convertible, A-1
  non-convertible, A-2
Address allocation, 4-4
Address areas, overview, 3-32
Address changes, 7-2
Address register, 3-45
Addressing
  absolute, 3-39
  data addresses, 3-41
  indirect, 3-43
    converting, 7-4
  memory-indirect, 3-44
  register-indirect, 3-45
  symbolic, 3-39
Analog functions, 3-29
Analog value processing, example, 9-2
ANY pointer, 9-12
AS-i, 2-10
AS511, 2-3
ASCII source file, 3-16
Assignment list, 3-39, 6-1, 6-4
Authorization, 3-2

B

Background processing, 3-20
Backup battery, 2-7
Basic functions, 3-29
Battery failure, 3-22
Bit logic instructions, 3-35
Bit memory, CPU, 2-6, 2-7
Block instructions, 3-37
Block transfer, 7-5
  example, 9-11
Block transfers, 3-37
Block types, in S5 and S7, 3-25
Blocks
  comparison STEP 5/STEP 7, 3-17
  CPU, 2-6, 2-7
Blocks container, STEP 7 object, 3-6
BR register, 7-5

C

Cam control, 2-13
CD-ROM, 2-1
Command output instructions, 3-37
Comment block, 3-17
Communication, event-driven, 2-19
Communication functions, 2-18
Communications processors, 2-10
Comparison instructions, 3-36
Compiling, 8-1
Complete restart, 3-20
Configuration tool, 2-22
Configuring
  communication connections, 3-11
  hardware, 3-9
Connection, configured to S5 station, 3-12
Connection table, 3-11
  STEP 7 object, 3-6
Consistency check, 8-1
Constant format, 3-31
Controller module, 2-13
Conversion, requirements, 4-2
Conversion instructions, 3-36
Convertible
  address, A-1
  instruction, A-3, A-4
COROS, 2-3
Counter instructions, 3-35
Counter module, 2-13
Counters, CPU, 2-6, 2-7
CP modules, 2-10

CPU, 5-3
analog inputs, 2-6, 2-7
analog outputs, 2-6, 2-7
bit memory, 2-6, 2-7
blocks, 2-6, 2-7
counters, 2-6, 2-7
DBs, 2-6, 2-7
digital inputs, 2-6, 2-7
digital outputs, 2-6, 2-7
FBs, 2-6, 2-7
FCs, 2-6, 2-7
load memory, 2-6, 2-7
local data, 2-6, 2-7
OBs, 2-6, 2-7
process image, 2-6, 2-7
retentive data, 2-6
S7-300, 2-6
S7-400, 2-7
SDBs, 2-7
SFBs, 2-6, 2-7
SFCs, 2-6, 2-7
timers, 2-6, 2-7
work memory, 2-6, 2-7
CRC, 3-23
Creating software, 3-13
inserting components, 3-15
Cross-reference list, 6-1
Cycle monitoring time, 3-23
Cyclic interrupt, 3-20

D

Data block (DB), 3-17, 3-18
Data block instructions, 3-36
DB. *See* Data block
DB register, 3-41, 3-42
DB1, 3-26
DB1/DX0, 4-4, 5-4
Diagnostic buffer, 2-15
Diagnostic interrupt, 2-15, 9-2
DIL switches, 2-5
Distributed I/O, 2-17
DP master, modules, 2-17
DP slave, modules, 2-17
DX. *See* Data block
DX0, 3-26

E

Edge change, 2-15
Error handling, 3-21
Error messages, 6-8
ET 200, 2-17
Ethernet, 2-10
Example
analog value processing, 9-2
block transfer, 9-11
start information, 9-8
temporary local data, 9-5
Expansion rack, 2-9

F

FB. *See* Function block
FC. *See* Function
FDL (SDA), 2-18
File formats, 3-40
Floating-point math, 3-28
Floating-point math instructions, 3-36
FM modules, 2-13
FMS master, 2-17
FMS service, 2-19
FMS slaves, 2-17
Fully integrated automation, 1-1
Function (FC), 3-17, 3-18
Function block (FB), 3-17, 3-18
Function modules, 2-13
FX. *See* Function block

G

GD communication. *See* Global data communication
Global data communication, 2-19

H

Handling block, 2-20
Hardware, STEP 7 object, 3-5
Hardware interrupt, 2-15, 3-20
HMI (Human Machine Interface), 2-3, 2-21

I

IM modules, 2-9
Importing
ASCII source file, 3-16
symbol table, 3-40
Indirect addressing, converting, 7-4

Industrial Ethernet, 2-10, 2-18
interface in user program, 2-20
modules, 2-11
Inputs
analog, 2-6, 2-7
digital, 2-6, 2-7
Installation, STEP 7 software, 3-2
Instruction
convertible, A-3, A-4
non-convertible, A-6
Instruction macro, 5-6
Instructions, overview, 3-35
Integer math instructions, 3-36
Interface modules, 2-9
Interprocessor communication flags, 3-23
Interrupt, 3-20, 3-22
Interrupt commands, 3-37
IP modules, 2-13
ISO transport, 2-18
ISO-on-TCP, 2-18

J

Jump instructions, 3-37

L

LIR, 4-3
Load instructions, 3-35
Load memory
CPU S7-300, 2-6
CPU S7-400, 2-7
Local data, 3-33
CPU, 2-6, 2-7

M

Macro, 5-5
creating, 5-8
Math functions, 3-29, 3-38
Memory, 4-3
Module catalog, 3-10
Module information, 5-3
Module parameters, comparison S5/S7, 2-5
Modules, overview, 2-4
MPI, 2-3, 2-10, 2-18
Multicomputing interrupt, 3-20
Multipoint interface, 2-3

N

Network, STEP 7 object, 3-5
Non-convertible
address, A-2
instruction, A-6
Null instructions, 3-38

O

OB. *See* Organization block
OB macro, 5-7
OB1, example, 9-14
Operator control and monitoring, 2-21
Operator Panel (OP), 2-21
Organization block (OB), 3-17, 3-20
Outputs
analog, 2-6, 2-7
digital, 2-6, 2-7

P

Page commands, 3-38
PB. *See* Program block
Performance, 2-2
PG interface, 2-10
Point-to-point connection, 2-10
interface in user program, 2-20
modules, 2-12
Pointer format, 3-43
Position detection modules, 2-13
Positioning module, 2-13
Power supply modules, 2-8
Process image, CPU, 2-6, 2-7
Processing functions, (DO FW, DO DW), 4-3
PROFIBUS, 2-3, 2-10, 2-18
interface in user program, 2-20
modules, 2-11
Program block (PB), 3-17
Programmable controllers, overview, 2-2
Programmable modules, 3-6
Programming device interface
AS511, 2-3
MPI, 2-3
Project, 3-4
creating, 3-7
Project file, 3-4
Proportioning module, 2-13
ProTool, 2-22

R

Register instructions, 3-35
Restart, 3-20
RET_VAL, 9-3
Retentive behavior, 4-4
Retentive data, CPU, 2-6
Retentivity, 2-7
Return value
  of a function, 9-3
  of a system function, 3-22
Rewire, 5-4, 7-2
Rotate instructions, 3-36

S

S5 expansion unit, 2-9
S5 handling block, 2-20
S5 standard function blocks, 7-6
S7 blocks, creating, 3-15
S7 project, creating, 4-4
SB. *See* Sequence block
Scratchpad flags, 3-33, 9-6
SDB. *See* System data block
Sequence block (SB), 3-17
Set/read CPU time, 3-22
SFB. *See* System function block
SFC. *See* System function
Shift instructions, 3-36
Shift register, 3-23
Signal functions, 3-28
Signal modules, 2-15
Signal preprocessing modules, 2-13
SIMATIC Manager, 3-3
  window, 3-13
SIMATIC S7, overview, 2-2
Simulator module, 2-16
SINEC H1, 2-11
SINEC L1, 2-11, 3-26
SINEC L2, 2-11, 3-26
SINEC S1, 2-11
SM modules, 2-15
Software, overview of components, 3-14
Source file, STEP 7 object, 3-6
Special functions, 3-22
Special OBs, 3-17
Standard functions, 3-28
Standard library, 3-15
Start information, 3-34, 9-8
Startup, 3-20
Station, STEP 7 object, 3-5
STEP 5 block, 3-17
STEP 5 project, 3-4
STEP 7
  installing, 3-2
  starting, 3-3
STEP 7 project, 3-4
  archiving, 3-8
  components, 3-5
  creating, 3-7
  storing, 3-8
STL compiler, 8-1
Stop instructions, 3-37
Subnet, 2-10
Symbol, local, 3-40
Symbol table, 3-40
  creating, 3-15
  example, 9-14
  STEP 7 object, 3-6
System data block (SDB), 3-17, 3-19
System function (SFC), 3-17, 3-19
System function block (SFB), 3-17, 3-19
System settings S5, 3-26

T

Time-delay interrupt, 3-20
Time-of-day interrupt, 3-20
Timer instructions, 3-35
Timers, CPU, 2-6, 2-7
TIR, 4-3
Tool, hardware conversion, 2-1
Transfer instructions, 3-35

U

User authorization, 3-2

V

Visualization, 2-22

W

Warning, Converter messages, 6-10
WinCC, 2-22
Word logic instructions, 3-36
Work memory, CPU, 2-6, 2-7

Siemens AG
A&D AS SM ID

Östliche Rheinbrückenstr. 50
D-76181 Karlsruhe
Federal Republic of Germany

From:

Your Name: _

Your Title: _

Company Name: _

Street: _

City, Zip Code _

Country: _

Phone: _

Please check any industry that applies to you:

- ❒ Automotive
- ❒ Chemical
- ❒ Electrical Machinery
- ❒ Food
- ❒ Instrument and Control
- ❒ Nonelectrical Machinery
- ❒ Petrochemical
- ❒ Pharmaceutical
- ❒ Plastic
- ❒ Pulp and Paper
- ❒ Textiles
- ❒ Transportation
- ❒ Other _ _ _ _ _ _ _ _ _ _ _

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Please give each of the following questions your own personal mark within the range from 1 (very good) to 5 (poor).

1. Do the contents meet your requirements? ☐
2. Is the information you need easy to find? ☐
3. Is the text easy to understand? ☐
4. Does the level of technical detail meet your requirements? ☐
5. Please rate the quality of the graphics/tables: ☐

Additional comments:

_ _ _ _